Fast Computational Algorithms for the Discrete Wavelet Transform and Applications of Localized Orthonormal Bases in Signal Classification

Eirik Fossgaard

Author address:

Department of mathematics, Faculty of Science,
University of Tromsø, 9037 Tromsø.
E-mail address: eirikf@math.uit.no

Abstract. In the first part of this paper we construct an algorithm for implementing the discrete wavelet transform by means of matrices in $SO_2(\mathbf{R})$ for orthonormal compactly supported wavelets and matrices in $SL_m(\mathbf{R}), m \geq 2$, for compactly supported biorthogonal wavelets. We show that in 1 dimension the total operation count using this algorithm can be reduced to about 50% of the conventional convolution and downsampling by 2-operation for both orthonormal and biorthogonal filters. In the special case of biorthogonal symmetric odd-odd filters, we show an implementation yielding a total operation count of about 38% of the conventional method. In 2 dimensions we show an implementation of this algorithm yielding a reduction in the total operation count of about 70% when the filters are orthonormal, a reduction of about 62% for general biorthogonal filters, and a reduction of about 70% if the filters are symmetric odd-odd length filters. We further extend these results to 3 dimensions.

In the second part of the paper we show how the $SO_2(\mathbf{R})$-method for implementing the discrete wavelet transform may be exploited to compute short FIR filters, and we construct edge mappings where we try to improve upon the preservation of regularity due to conventional methods.

In the third part of the paper we consider the problem of discriminating two classes of radar signals generated from some number of point sources distributed randomly in a bounded plane domain. A statistical space-frequency analysis is performed on a set of training signals using the LDB-algorithm of N.Saito and R.Coifman. In this analysis we consider several dictionaries of orthonormal bases. The resulting most discriminating basis functions are used to construct classifiers. The success of different dictionaries is measured by computing the misclassification rates of the classifiers on a set of test signals.

Fast Computational Algorithms for the Discrete Wavelet Transform and Applications of Localized Orthonormal Bases in Signal Classification

Eirik Fossgaard

january 1999

University of Tromsø

1999

Author	:	Eirik Fossgaard
Publisher	:	University of Tromsø
ISBN	:	82-90487-93-2

Contents

Acknowledgements

I want to express my special thanks to my advisor, Professor Jan-Olov Strömberg at Tromsø University Math Department, for inviting me to come with him during his research year at Yale University Math Department 1995/1996, and for his helpful and patient guidance through my work with this paper.

I thank Yale University Math Department and especially Professor R.R. Coifman for providing space for me during my stay there.

And finally, but not least, I thank my fellow students at the Faculty of Mathematical Sciences at Tromsø University for many helpful discussions and for being exceptionally social humans.

CHAPTER 1

Preliminaries and Concepts

1. A library of bases for $L^2(\mathbf{R})$

Let $L^2(\mathbf{R})$ denote the space of all square integrable functions of a single real variable. We present some well-known bases for this space.

1.1. Wavelet bases. Expressed shortly, these bases consist exclusively of all dyadic dilations and integer translations of a single *mother wavelet* ψ with zero integral, that is

$$f \in L^2(\mathbf{R}) \Longrightarrow f = \sum_{j,k\in\mathbf{Z}} d_{j,k}\psi_{j,k} \text{ where } \psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k),$$

with the series converging in L^2 sense. In this paper we will only be concerned with real and compactly supported wavelets. In practice, ψ will possess some localization in both time and frequency, and the basis will have to be stable, thus asserting the existence of positive numbers A and B such that

$$A\|f\|^2 \leq \sum_{j,k\in\mathbf{Z}} |d_{j,k}|^2 \leq B\|f\|^2,$$

and ψ will possess some *vanishing moments*, that is

$$\int \psi(x)x^l dx = 0, \quad 0 \leq l \leq M, \ M \geq 1.$$

We can further separate this family of bases into two subfamilies:

- *Orthonormal Wavelet Bases.* The mother wavelet ψ is orthonormal to all its dyadic dilations and integer translates, thus asserting

$$\langle \psi_{j,k}, \psi_{j',k'} \rangle = \delta_{j,j'}\delta_{k,k'} \ , \ d_{j,k} = \langle f, \psi_{j,k} \rangle \, .$$

- *Biorthogonal Wavelet Bases.* The mother wavelet is biorthogonal to all dyadic dilations and integer translates of its dual $\widetilde{\psi}$, yielding

$$\left\langle \psi_{j,k}, \widetilde{\psi}_{j',k'} \right\rangle = \delta_{j,j'}\delta_{k,k'} \ , \ d_{j,k} = \left\langle f, \widetilde{\psi}_{j,k} \right\rangle .$$

The $\psi_{j,k}$ are called the analysis wavelets, while the $\widetilde{\psi}_{j,k}$ are called the synthesis wavelets.

Wavelet bases are intimately connected to the concept of a "Multiresolution Analysis" (MRA).

DEFINITION 1.1. *A Multiresolution Analysis is a nested sequence* $\{V_j\}_{j\in\mathbf{Z}}$ *of closed subspaces of* $L^2(\mathbf{R})$ *satisfying*

1. $V_j \subset V_{j+1}$, $j \in \mathbf{Z}$.
2. $\overline{\bigcup_{j\in\mathbf{Z}} V_j} = L^2(\mathbf{R})$.
3. $\bigcap_{j\in\mathbf{Z}} V_j = \{0\}$.
4. $f(x) \in V_j \Longrightarrow f(x-k) \in V_j$, $k \in \mathbf{Z}$.
5. $f(x) \in V_j \Longrightarrow f(2x) \in V_{j+1}$.
6. *There exists a "scaling function"* $\phi \in V_0$ *such that* $\{\phi_{j,k}\}_{k\in\mathbf{Z}}$ *constitute a Riesz basis (= stable basis) for* V_j , $j \in \mathbf{Z}$.

The great triumph of MRA is the following result which proof can be found in [**1**]:

THEOREM 1.1. *Given a MRA, there exists a wavelet* $\psi \in V_1 \cap V_0^c$ *such that* $\overline{\mathrm{Span}_{j,k}\,\psi_{j,k}} = L^2(\mathbf{R})$. *The wavelet can be chosen to have compact support.*

Remark. The wavelets possessing the maximum number of vanishing moments compatible with their support width, are called Daubechies' wavelets. It is shown in [**1**] that a wavelet with N vanishing moments has support width $2N-1$. It is also possible to assign vanishing moments to the scaling function ϕ, (except a zeroth vanishing moment). Wavelets where both the wavelet and the corresponding scaling function have vanishing moments are called coiflets.

From the first MRA property we get the "scaling iteration equation"

$$\phi_{j,k} = 2^{1/2} \sum_{l\in\mathbf{Z}} H(l-2k)\phi_{j+1,l} \ , \ H(l) = \langle \phi, \phi_{1,l} \rangle \, . \tag{1.1}$$

If the $\{\phi_{0,k}\}_{k\in\mathbf{Z}}$ form an orthonormal basis for V_0 (if not we can carry out an "orthonormalization trick", see [**1**] for details) we get an orthonormal wavelet basis from the MRA by defining the *wavelet* space W_j at scale j as the orthogonal complement of V_j in V_{j+1}, that is

$$V_{j+1} = W_j \oplus V_j. \tag{1.2}$$

Now, the properties of the scaling spaces V_j together with the definition (1.2) yields the *orthogonal* decomposition

$$L^2(\mathbf{R}) = \bigoplus_{j\in\mathbf{Z}} W_j, \tag{1.3}$$

and the "wavelet scaling equation"

$$\psi_{j,k} = 2^{1/2} \sum_{l \in \mathbf{Z}} G(l-2k)\phi_{j+1,l} \ , \ G(l) = \langle \psi, \phi_{1,l} \rangle \, . \tag{1.4}$$

The vanishing integral of ψ implies a non-vanishing integral for ϕ. To see this, assume $\int \phi = 0$ and consider the characteristic function $\chi_{[-K,K]} \in L^2$. By choosing K large enough, we get a contradiction to the fact that $\{\phi_{j,k}\}_{j,k \in \mathbf{Z}}$ constitute a Riesz basis for L^2.

Since $\sum_n H(n) = \int \phi \neq 0$ and $\sum_n G(n) = \int \psi = 0$, the $H(n)$ make up a *lowpass filter*, convolving it with $\mathbf{x} \in \mathbf{R}^n$ produces weighted local averages, while the g_n make up a *highpass filter*, convolving it with $\mathbf{x} \in \mathbf{R}^n$ produces details by catching up local oscillations. Filters with finite support width (i.e finite number of nonzero coefficients) are called "finite impulse response filters" or simply FIR filters.

By multiplying equation (1.1) on both sides by $\phi_{j,k}$ and integrating the result, using orthonormality of ϕ to its integer translates, we get orthonormality of the lowpass filter to its even translates

$$\delta_{0,k} = \sum_{n \in \mathbf{Z}} H(n)H(n+2k). \tag{1.5}$$

In [**1**] it is shown that the highpass coefficients $G(n)$ are uniquely determined by the lowpass coefficients $H(n)$ up to

$$G(n) = (-1)^n H(2N+1-n) \text{ , modulo phase and } N \in \mathbf{Z}. \tag{1.6}$$

The finite support of the wavelet ψ and the filters $\{H(k)\}_{k \in \mathbf{Z}}$, $\{G(k)\}_{k \in \mathbf{Z}}$ are thus seen to be simple consequences of the finite support of the scaling function. We can also see that by relation (1.6) the lowpass and highpass filters have the same length, moreover, it has to be *even* by relation (1.5).

We remark that by Fourier transforming equation (1.1) and iterating the result, we get a (infinite) product formula for $\widehat{\phi}$, given by

$$\begin{aligned} \hat{\phi}(\xi) &= \frac{1}{\sqrt{2\pi}} \prod_{j=1}^{\infty} m_0(2^{-j}\xi) \\ m_0(\xi) &= \frac{1}{\sqrt{2}} \sum_{n=0}^{L-1} H(n) e^{-i \cdot n\xi}, \end{aligned}$$

thus ϕ (and consequently ψ) is uniquely determined by the filter coefficients $H(n)$.

Now, given the projection $P_J f$ of a function f on some scaling space V_J, there is an easy way of computing the projection of f on all "coarser" spaces $\{V_j, W_j\}_{j<J}$ without integrating. From (1.1) and (1.4) we obtain a set of recursion relations:

$$\langle f, \phi_{j-1,k} \rangle = 2^{1/2} \sum_l H(l-2k) \langle f, \phi_{j,l} \rangle , \tag{1.7}$$

$$\langle f, \psi_{j-1,k} \rangle = 2^{1/2} \sum_l G(l-2k) \langle f, \phi_{j,l} \rangle . \tag{1.8}$$

We observe that at each scale j the projection $P_j f$ is decomposed into two parts:

- *Local averages.* The lowpass coefficients $c_k^j \equiv \langle f, \phi_{j,k} \rangle$ are the result of passing $\{c_l^{j-1}\}_{l \in \mathbf{Z}}$ through the "lowpass channel" determined by (1.7). This operation can be described as "convolution and downsampling by 2": First convolve with $\{H(-k)\}_{k \in \mathbf{Z}}$, then throw away the odd numbered indicies. This operation is often called "lowpass filtering". With the convention $c = \{c_n\}_{n \in \mathbf{Z}}$, we can write a (infinite) matrix equation

$$c^{j-1} = \mathbf{H} c^j, \ \mathbf{H}(i,j) = H(i-2j), \ 0 \le i, j < \infty. \tag{1.9}$$

- *Local oscillations.* Similarly, the highpass coefficients d_k^j result from passing $\{c_l^{j-1}\}_{l \in \mathbf{Z}}$ through the highpass channel determined by (1.8) and is called "highpass filtering". We write

$$d^{j-1} = \mathbf{G} c^j \ , \mathbf{G}(i,j) = G(i-2j), \ 0 \le i, j < \infty. \tag{1.10}$$

The decomposition of $P_J f$ into averages and details at different scales is illustrated in Figure 1. This decomposition is *the discrete wavelet transform* in dimension 1.

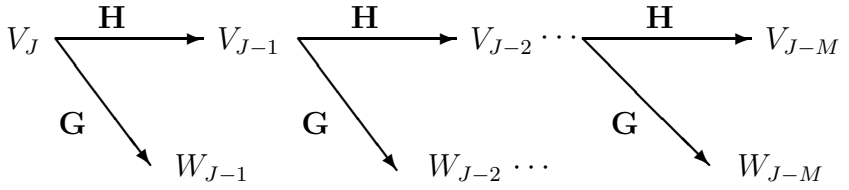

FIGURE 1. The "logarithmic tree" of a wavelet decomposition in M levels.

The inverse operation is in this orthogonal case given by the transpose matrices:

$$\begin{aligned} c_k^j &= \mathbf{H}^t c^{j-1}(k) + \mathbf{G}^t d^{j-1}(k) \\ &= 2^{1/2} \sum_{l \in \mathbf{Z}} H(k-2l) c_l^{j-1} + 2^{1/2} \sum_{l \in \mathbf{Z}} G(k-2l) d_l^{j-1}. \end{aligned} \tag{1.11}$$

This operation is illustrated by simply reversing the arrows in Figure 1 and replacing $\mathbf{H}, \mathbf{G}$ by their transposes.

We emphasize that the expansion of a function into a wavelet basis is fast: From the equations (1.7) and (1.8) we get that a complete wavelet analysis of a sequence of length N has a total operation cost of no more than $4L \cdot N$ multiply-adds, where L is the support width of the filters. Equation (1.11) yields the same bound on the operation cost for reconstruction.

In summary the FIR filters $\{H(n)\}_{n\in\mathbf{Z}}$, $\{G(n)\}_{n\in\mathbf{Z}}$ make up a two-channel orthogonal filter bank with perfect reconstruction.

If ϕ is not orthogonal to its translates, the sums in (1.2) and (1.3) are *direct* rather than orthogonal and the wavelet basis is no longer its own dual. The construction of a *biorthogonal* Riesz wavelet basis is shown in [**1**]. Thus we get two MRA's: The analysis MRA that is built up from the analysis scaling functions $\phi_{j,k}$ and the corresponding wavelets $\psi_{j,k}$, and the synthesis MRA built from the synthesis scaling functions $\widetilde{\phi}_{j,k}$ and their wavelets $\widetilde{\psi}_{j,k}$.

The relations between the basis functions and filters from the two MRA's are given below. A simple picture of the biorthogonality relations is shown in Figure 2.

Filter relations:

$$\delta_{0,k} = \sum_n H(n)\widetilde{H}(n+2k)\ , \tag{1.12}$$

$$G(n) = (-1)^{n+1}\widetilde{H}(2N+1-n),$$
$$\widetilde{G}(n) = (-1)^{n+1}H(2N+1-n), N \in \mathbf{Z}. \tag{1.13}$$

Biorthogonality relations:

$$\left\langle \psi_{j,k}, \widetilde{\psi}_{j',k'} \right\rangle = \delta_{j,j'}\delta_{k,k'} \Longleftrightarrow \left\langle \phi_{0,k}, \widetilde{\phi}_{0,k'} \right\rangle = \delta_{k,k'}. \tag{1.14}$$

The scaling equations for the biorthogonal scaling- and wavelet functions and the expansion and reconstruction equations are immediate generalizations of the corresponding equations in the orthogonal case. We note that the lengths of the filters H and $\tilde{H}$ need not be equal.

When working in $L^2(\mathbf{R}^n)$, that is with square integrable functions of n variables, it is possible to generate a wavelet basis for this space by taking tensor products of n (possibly different) MRA's in $L^2(\mathbf{R})$. This results in $2^n - 1$ different mother functions, the dilations and translates of which make up what we call a *tensor* wavelet basis in dimension n. The iteration is done on the *pure* lowpass band, that is the coefficients originating from the *pure* tensor scaling function $\phi(x_1) \otimes \phi(x_2) \otimes \cdots \otimes \phi(x_n)$.

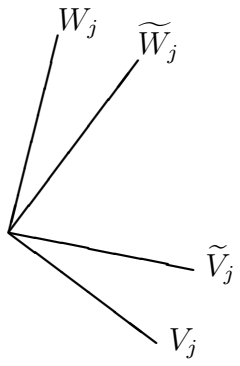

FIGURE 2. The scaling- and wavelet spaces V_j, W_j and their duals $\widetilde{V}_j$, $\widetilde{W}_j$.

1.2. Wavelet packet bases. These bases are particular linear combinations of elements in a wavelet basis. It follows that the elements in a wavelet packet basis inherit some of the properties of the wavelets they are made of, such as compact support and some localization in both time and frequency. We will see that the wavelet basis is included as a special case of wavelet packet bases.

Following [**2**], we briefly describe the construction of these bases. Define

$$\begin{aligned}
\psi_0 = H\psi_0; \qquad & \int_{\mathbf{R}} \psi_0 = 1, \\
\psi_{2n} = H\psi_n; \qquad & \psi_{2n}(t) = 2^{1/2} \sum_{j\in\mathbf{Z}} H(j)\psi_n(2t-j), \\
\psi_{2n+1} = G\psi_n; \qquad & \psi_{2n+1}(t) = 2^{1/2} \sum_{j\in\mathbf{Z}} G(j)\psi_n(2t-j), \\
\psi_{s,f,p}(t) = 2^{-s/2}\psi_f(2^{-s}t-p); \qquad & \Lambda_f = \overline{\operatorname{Span}_p \psi_{0,f,p}}, \\
\sigma x(t) = 2^{-1/2}x(t/2); \qquad & \sigma\Lambda_f = \{\sigma x : x \in \Lambda_f\},
\end{aligned} \tag{1.15}$$

where H, G is a pair of conjugate FIR filters and $\psi_{s,f,p}$ is called a wavelet packet of scale index s, frequency index f and position index p. We immediately have that $\sigma^s\Lambda_f$ is the closure of $\operatorname{Span}_p \psi_{s,f,p}$. Exploiting the natural one-to-one correspondence

$$I_{s,f} \longleftrightarrow \sigma^s\Lambda_f; \qquad I_{s,f} = \left[\frac{s}{2^f}, \frac{s+1}{2^f}\right),$$

we have the following result for which proof we refer to [**2**]:

THEOREM 1.2. *If $\mathcal{I}$ is a dyadic cover of $\mathbf{R}^+$, then $\{\overline{\bigcup_{s,f} \sigma^s\Lambda_f} : I_{s,f} \in \mathcal{I}\} = L^2(\mathbf{R})$ and if $\mathcal{I}$ is disjoint, the wavelet packets*

$\{\psi_{s,f,p} : I_{s,f} \in \mathcal{I}, p \in \mathbf{Z}\}$ form a basis for $L^2(\mathbf{R})$. Furthermore, if the filters H, G are orthogonal, this basis is an orthonormal basis.

To keep the same filtering formulas for sequences as for functions, make the definitions:

$$\psi^{<}_{s,f,p} = 2^{-s/2}\psi_f(p - 2^{-s}t),$$
$$\lambda_{s,f,p} = \left\langle x, \psi^{<}_{s,f,p} \right\rangle, \ x \in \sigma^s \Lambda_f.$$

Then it is easy to verify the following recursion relations:

$$\lambda_{s+1,2f,p} = H\lambda_{s,f,p},$$
$$\lambda_{s+1,2f+1,p} = G\lambda_{s,f,p}. \tag{1.16}$$

Thus, identifying $\sigma^s \Lambda_f$ with $\Omega_{s,f}$, the expansion of a function into a collection of wavelet packet bases can be illustrated as in Figure 3. We note that every disjoint cover of the top box in this figure by smaller boxes from levels below, yields a new basis for the space $\Omega_{0,0}$.

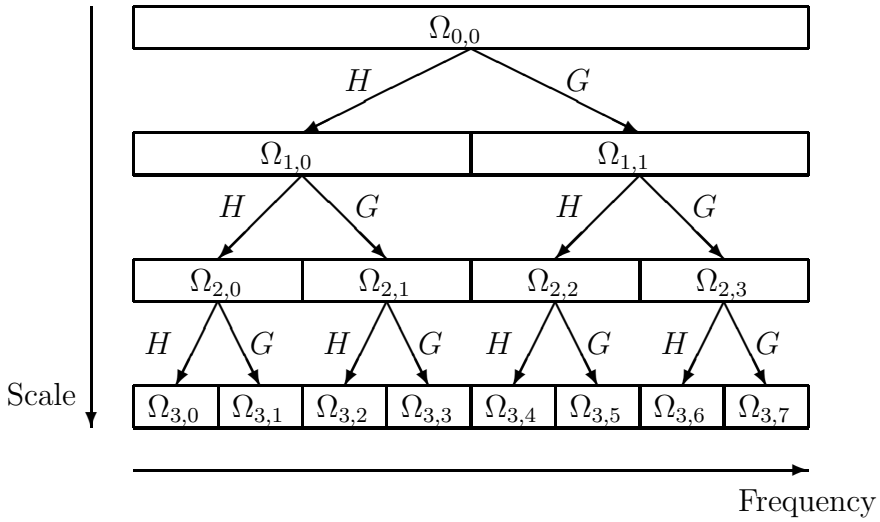

FIGURE 3. The binary tree of a wavelet packet decomposition in 4 levels.

By comparing Figure 1 and Figure 3 it is easy to see that the logarithmic tree of expansion into a wavelet basis is a subtree of the binary tree of expansion into a collection of wavelet packet bases. We conclude that the wavelet basis is included in the collection of wavelet packet bases.

For reconstruction one needs the dual basis of the particular wavelet packet basis. It is shown in [**2**] that the wavelet packets defined by the dual filters H', G' are the duals of the wavelet packets defined by H and G.

Given a sequence of length N, a rather crude estimate given in [**2**] shows that a wavelet packet analysis of the sequence will provide more than 2^N different bases. The recursion relations (1.16) makes the expansion into a wavelet packet analysis cheap: The total operation cost will be no more than $L \cdot N \log_2 N$ multiply-adds, where L is the support width of the filters.

1.3. Smooth local trigonometric bases. The huge drawback of the Fourier basis for applications in signal analysis is its "non-locality" in time, thus it is impossible to relate specific frequencies of a signal to specific time or space windows using this basis. Also, this basis only span the space of the periodic $L^2(\mathbf{R})$ functions. What we would like is a trigonometric basis for $L^2(\mathbf{R})$ with the following desirable properties:

- Orthogonality.
- Smoothness: The basis elements should possess at least a few continuous derivatives.
- Localization in space: Each basis element should have finite support.
- Efficiency: There should exist a fast algorithm (i.e. $N \log_2 N$) for expansion into the basis.

Such bases do exist, we refer to [**2**] for a thorough exposition on the subject. We will briefly outline the construction of a Local Sine Basis and prove that this basis enjoys the properties above. Define the bell functions $\{b_k(x)\}_{k\in\mathbf{Z}}$ with the following properties:

$$\begin{aligned} &\sum_{k\in\mathbf{Z}} b_k^2(x) = 1, \ \forall x \in \mathbf{R}, \\ &supp(b_k) = I_k = (\alpha_k - \epsilon_k, \alpha_{k+1} + \epsilon_{k+1}), \\ &b_k b_{k+2} = 0; \ b_{k-1}b_k \text{ is even around } \alpha_k. \end{aligned} \tag{1.17}$$

A schematic picture of these bell functions is shown in Figure 4.

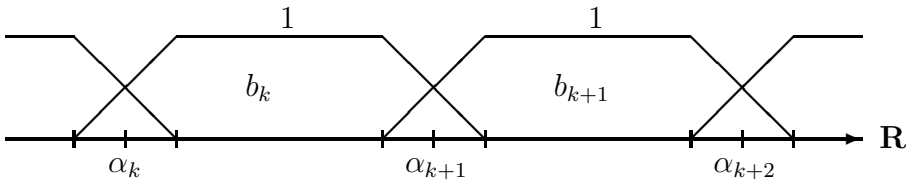

FIGURE 4. The bell functions.

Define

$$s_{j,k}(x) = \frac{b_k(x)}{(\alpha_{k+1} - \alpha_k)^{1/2}} \sin\left(\pi(j + 1/2)\frac{x - \alpha_k}{\alpha_{k+1} - \alpha_k}\right). \tag{1.18}$$

Then we have the following result:

THEOREM 1.3. *The $s_{j,k}$ form a smooth orthogonal basis for $L^2(\mathbf{R})$. Furthermore, this basis enjoys all the desirable properties listed above.*

Proof: The smoothness and localization properties of this basis are immediately clear. To prove orthogonality we only need to consider $\langle s_{j,k}, s_{j',k'} \rangle$ when $|k - k'| \leq 1$ because of the support property of the bell functions. Define

$$\widetilde{s}_{j,k}(x) = \frac{1}{(\alpha_{k+1} - \alpha_k)^{1/2}} \sin\left(\pi(j + 1/2)\frac{x - \alpha_k}{\alpha_{k+1} - \alpha_k}\right).$$

For $k - k' = 1$ we get

$$\langle s_{j,k}, s_{j',k-1} \rangle = \int_{\alpha_k - \epsilon_k}^{\alpha_k + \epsilon_k} b_{k-1}(x) b_k(x) \widetilde{s}_{j',k-1}(x) \widetilde{s}_{j,k}(x) dx = 0.$$

To see this, observe that since $b_{k-1}b_k$ is even around α_k, the integrand is an odd function around the center of integration, thus the integral equals zero. For $k = k'$ we get

$$\begin{aligned} \langle s_{j,k}, s_{j',k} \rangle &= \int_{\alpha_k - \epsilon_k}^{\alpha_{k+1} + \epsilon_{k+1}} b_k^2(x) \widetilde{s}_{j',k}(x) \widetilde{s}_{j,k}(x) dx \\ &= \int_{\alpha_k}^{\alpha_{k+1}} \widetilde{s}_{j',k}(x) \widetilde{s}_{j,k}(x) dx = \delta_{j,j'}. \end{aligned}$$

To see that the first integral equals the second, observe that $\widetilde{s}_{j',k}\widetilde{s}_{j,k}$ is even around α_k and $b_k^2(x) = 1 - b_k^2(\alpha_k + \epsilon_k - x),\ x \in [\alpha_k - \epsilon_k, \alpha_k + \epsilon_k]$. The second integral is an elementary calculation. This shows that the $s_{j,k}$ are orthonormal over all frequencies j and translates k.

Now, let $f \in L^2(\mathbf{R})$ and consider $\langle f, s_{j,k} \rangle$. We split the integral into three parts and change variables on the leftmost and rightmost integrals to obtain a formula for the inner product involving integration only on the interval $[\alpha_k, \alpha_{k+1}]$. We may express this operation by saying that the part of f that lives in $[\alpha_k - \epsilon_k, \alpha_k] \cup [\alpha_{k+1}, \alpha_k + \epsilon_{k+1}]$ is folded into the interval $[\alpha_k, \alpha_{k+1}]$.

$$\begin{aligned}
\langle f, s_{j,k}\rangle &= \int_{\alpha_k-\epsilon_k}^{\alpha_{k+1}+\epsilon_{k+1}} f(x)b_k(x)\widetilde{s}_{j,k}(x)dx \\
&= \int_{\alpha_k-\epsilon_k}^{\alpha_k}(\cdot) + \int_{\alpha_k}^{\alpha_{k+1}}(\cdot) + \int_{\alpha_{k+1}}^{\alpha_{k+1}+\epsilon_{k+1}}(\cdot) \\
&= \int_{\alpha_k}^{\alpha_{k+1}} b_k(x)f(x)\widetilde{s}_{j,k}(x)dx \\
&- \int_{\alpha_k}^{\alpha_{k+1}} b_k(2\alpha_k - x)f(2\alpha_k - x)\widetilde{s}_{j,k}(x)dx \\
&+ \int_{\alpha_k}^{\alpha_{k+1}} b_k(2\alpha_{k+1} - x)f(2\alpha_{k+1} - x)\widetilde{s}_{j,k}(x)dx \\
&= \int_{\alpha_k}^{\alpha_{k+1}} f^\sharp(x)\widetilde{s}_{j,k}(x)dx, \\
f^\sharp(x) &\equiv b_k(x)f(x) - b_k(2\alpha_k - x)f(2\alpha_k - x) \\
&+ b_k(2\alpha_{k+1} - x)f(2\alpha_{k+1} - x).
\end{aligned}$$

To implement the local sine basis, we place the gridpoints at half-integers, and evaluate the integral by $DST - IV$, meaning a discrete sine transform of type IV. This transform enjoys the nice property of being its own inverse, and has a $N \log_2 N$ implementation. We refer to [**2**] for a proof of these facts.

It only remains to prove completeness. We have $\langle f, s_{j,k}\rangle = \langle f^\sharp, \widetilde{s}_{j,k}\rangle$. Furthermore, $\mathrm{Span}_j\, \widetilde{s}_{j,k} = L^2([\alpha_k, \alpha_{k+1}))$ because of the completeness of the Fourier basis in $L^2([0, 2\pi))$. This yields

$$\begin{aligned}
f^\sharp\big|_{[\alpha_k,\alpha_{k+1})}(x) &= \sum_{j=0}^{\infty} \left\langle f^\sharp, \widetilde{s}_{j,k}\right\rangle \widetilde{s}_{j,k}(x) \\
&\Downarrow \\
b_k(x)f^\sharp(x) &= \sum_{j=0}^{\infty} \left\langle f^\sharp, \widetilde{s}_{j,k}\right\rangle b_k(x)\widetilde{s}_{j,k}(x) \\
&= \sum_{j=0}^{\infty} \langle f, s_{j,k}\rangle\, s_{j,k}(x) = P_k(f)(x), \\
P_k(f)(x) &\equiv b_k^2(x)f(x) \\
&- b_k(x)b_k(2\alpha_k - x)f(2\alpha_k - x) \\
&+ b_k(x)b_k(2\alpha_{k+1} - x)f(2\alpha_{k+1} - x).
\end{aligned}$$

Summing $P_k(f)$ over all $k \in \mathbf{Z}$, it is easy to see that all the terms cancel except the $b_k^2(x)f(x)$ terms. Thus we get

$$\sum_{k\in\mathbf{Z}}\sum_{j=0}^{\infty}\langle f, s_{j,k}\rangle\, s_{j,k}(x) = \sum_{k\in\mathbf{Z}} P_k(f)(x) = \left(\sum_{k\in\mathbf{Z}} b_k^2(x)\right) f(x) = f(x),$$

this confirms the completeness of the $s_{j,k}$. □

We note that by replacing sines by cosines and $DST-IV$ by $DCT-IV$ in the construction above, we get a local cosine basis for $L^2(\mathbf{R})$.

1.4. Local sine/cosine packet analysis. It is immediately clear how the notion of a wavelet packet analysis can be transfered to this case of local trigonometric bases. We restrict ourselves to bells of a fixed length at each level of decomposition. Thus, we replace each bell from the level above by two child bells of half the length of the parent bell. A schematic picture of this analysis is shown in Figure 5.

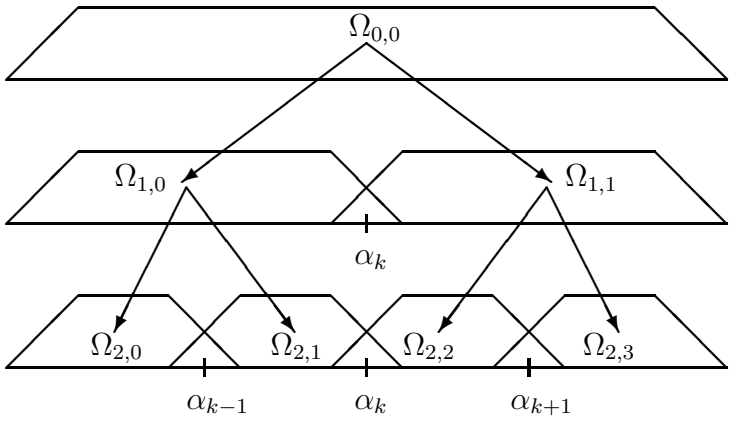

FIGURE 5. *Dyadic local sine/cosine analysis in three levels.*

It is easy to see that in analog to the case of wavelet packets, every cover of the original interval by our special bell functions corresponds to a (unique) basis for this interval. To get back to the original coordinates, simply perform the $DST - IV$ or $DCT - IV$ (they are their own inverses) on each subinterval, then "unfold" the result back into the folding regions $[\alpha_k - \epsilon_k, \alpha_k + \epsilon_k]$.

For a input of length N, the number of levels in a Local Sine/Cosine Analysis cannot exceed $\log_2 N$, thus the total operation cost is bounded by $N(\log_2 N)^2$ multiply-adds.

2. Signal classification using local feature extraction

The notation and ideas presented below are due to [**4**], to which we refer for a more thorough review of this subject. We say that a wavelet packet analysis or a local (co)sine packet analysis constitute a *dictionary* of bases for L^2. We call a collection of dictionaries a *library* of bases for L^2.

2.1. Local feature extraction. We define the problem of signal classification as the construction of the map c

$$c : \mathcal{S} \subset \mathbf{R}^n \longmapsto \mathcal{C} = \{1, 2, ..., C\},$$

where $\mathcal{S}$ is the set of all signals under consideration and $\mathcal{C}$ is the set of all relevant class names. We call $\mathcal{S}$ the *signal space*, $\mathcal{C}$ the *response space* and c a *classifier*. Since the dimension n of the signal space is normally very large compared to the dimension of the response space, it is important to extract only the relevant *features* of the signals to obtain an efficient and accurate classification. In other words, we want to map the information relevant to our problem into a few coordinates and ignore all the rest. To achieve this, define the map f

$$f : \mathcal{S} \longmapsto \mathcal{F} \subset \mathbf{R}^k, \ k \leq n,$$

$\mathcal{F}$ is called the *feature space* and f the *feature extractor*. Thus, once we have f, we only have to construct $c : \mathcal{F} \mapsto \mathcal{C}$, which in general is much easier because $k << n$.

To construct the feature extractor we use a training data set $\tau = \{\mathbf{s}_i, c_i\}_{i=1}^N \subset \mathcal{S} \times \mathcal{C}$ of N training signals $\mathbf{s}_i$ and their class names c_i. We will denote by N_i the number of training signals belonging to class i, so that $N = N_1 + \cdots + N_C$. Once f and c have been constructed, we measure the *misclassification rate* r_{error} using a test data set $\tau' = \{\mathbf{s}'_i, c'_i\}_{i=1}^{N'}$ which has not been used in the construction of f and c, by

$$r_{error} = \frac{1}{N'} \sum_{i=1}^{N'} \delta(c'_i - c(\mathbf{s}'_i)), \ \delta(0) = 0, \ \delta(x) = 1, \ x \neq 0,$$

We will use feature extractors on the form $f = \Theta^{(k)} \circ \Psi$ where $\Psi \in SO_2(n)$ with column vectors $\mathbf{w}_{j,k,l}^T$ with the correspondence

$$\begin{aligned} &\operatorname*{Span}_l \mathbf{w}_{j,k,l}^T = \Omega_{j,k}, \\ &j = 0, ..., J, \ k = 0, ..., 2^j - 1, \ l = 0, ..., 2^n - 1, \end{aligned} \tag{1.19}$$

where $\Omega_{j,k}$ are the subspaces of a wavelet packet or a local sine/cosine analysis. $\Theta^{(k)} : \mathcal{S} \longmapsto \mathcal{F}$ is a *selection rule* picking out the k most relevant coordinates from n coordinates.

Figure 6 illustrates two examples of $\mathcal{C} = \{1, 2\}$ using some Ψ and some $\Theta^{(2)}$. The classifiers c are indicated in the figure.

There are several possible choices for a classifier c, such as Linear Discriminant Analysis (LDA) and Classification and Regression Trees (CART), we refer to [**4**] for a review of these, as we will not make use of them in this paper.

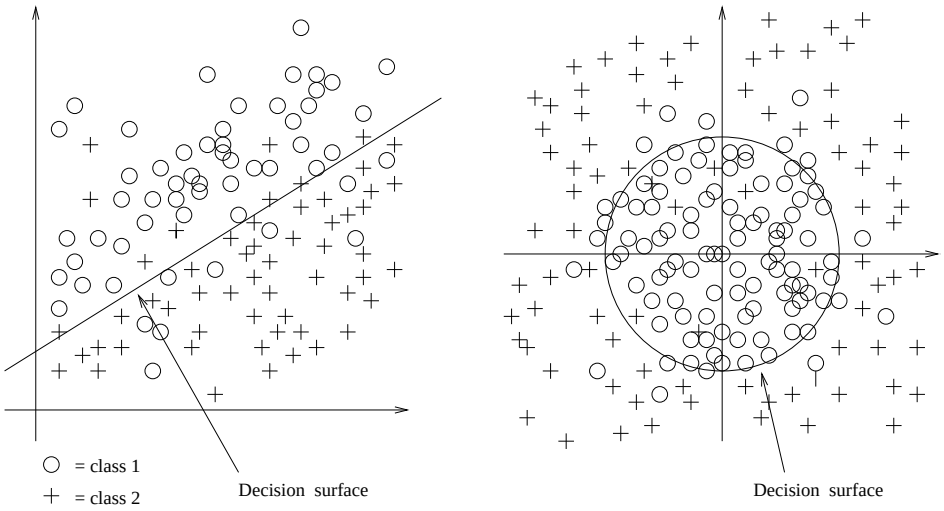

FIGURE 6. Example of sample plots of the two most discriminating coordinates in the two-class case.

2.2. Entropy and discriminant measures. Given a norm $\|\cdot\|_r$, we define the entropy H_r of a sequence $\mathbf{x}$ as

$$H_r(\mathbf{x}) = -\sum_i \frac{|x_i|^r}{\|\mathbf{x}\|_r^r} \log_2 \frac{|x_i|^r}{\|\mathbf{x}\|_r^r}, \ 1 \le r < \infty.$$

$H_r(\mathbf{x})$ measures the degree of order in the sequence $\mathbf{x}$, that is the information cost in describing $\mathbf{x}$.

In the two-class case, we need a *discriminant* measure d that measures how differently two sequences are distributed, in other words the relative entropy of the two sequences. In this paper we will only use $d = m_p$ defined as

$$m_p(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p^p = \sum_i (x_i - y_i)^p, \tag{1.20}$$

with $p = 2$ in most cases.

In the general case of C classes, we define the discriminant measure of C sequences as

$$d(\{\mathbf{x}^{(c)}\}_{c=1}^C) \equiv \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}). \tag{1.21}$$

We see that once we have a discriminant measure, we are capable of evaluating the power of discrimination of the different subspaces in any dictionary $\mathcal{D}$. This indicates that it is possible to choose a basis in $\mathbf{R}^n$ for the set $\mathcal{S}$ of signals with the property that no other basis in the dictionary will discriminate more between classes. Furthermore, the

discriminant measure should be *additive* to ensure a fast computational algorithm.

DEFINITION 1.2. *A discrimant measure d is said to be additive if* $d(\mathbf{x}, \mathbf{y}) = \sum_i d(x_i, y_i)$.

2.3. The local discriminant basis algorithm. Given a discriminant measure d, how do we best evaluate the power of discrimination in each subspace of a dictionary $\mathcal{D}$, and how do we select the most discriminating basis? The result should ideally only depend on the characteristic features of each class. The "local discriminant basis algorithm" (LDB-algorithm) developed in [**4**] yields a particular basis called a *local discriminant basis*, LDB for short, as described below.

DEFINITION 1.3. *Let $\{\mathbf{x}_i^{(c)}\}_{i=1}^{N_c}$ be a set of training signals belonging to class c. Then the time-frequency energy map of class c, Γ_c , is a table of real values specified by (j, k, l) as*

$$\Gamma_c(j,k,l) = \sum_{i=1}^{N_c} \left(\mathbf{w}_{j,k,l}^T \cdot \mathbf{x}_i^{(c)} \right)^2 / \sum_{i=1}^{N_c} \|\mathbf{x}_i^{(c)}\|^2, \tag{1.22}$$

for $j = 0, ..., J,\ k = 0, ..., 2^j - 1,\ l = 0, ..., 2^{n-j} - 1$.

For notational convenience, define

$$\begin{aligned}
&d(\{\Gamma_c(j,k,\cdot)\}_{c=1}^C) = \sum_{l=0}^{2^{n-j}-1} d(\Gamma_1(j,k,l), ..., \Gamma_C(j,k,l)),\\
&B_{j,k} = \operatorname*{Span}_{0 \le l \le 2^{n-j}-1} \mathbf{w}_{j,k,l} \subset \Omega_{j,k},\\
&A_{j,k} = \text{LDB}|_{B_{j,k}},\\
&\Delta_{j,k} = \text{ discriminant measure of } \Omega_{j,k}.
\end{aligned}$$

Then we have the following algorithm:

ALGORITHM 1.1. *(The Local Discriminant Basis Selection Algorithm). Given a training dataset τ consisting of C classes of signals $\{\{\mathbf{x}_i^{(c)}\}_{i=1}^{N_c}\}_{c=1}^C$,*

Step 0: *Choose a dictionary $\mathcal{D}$ of orthonormal wavelet packets or local sine/cosine packets and specify the maximum depth J of decomposition and an additive discriminant measure d.*

Step 1: *Construct time-frequency energy maps Γ_c for c = 1,...,C.*

Step 2: *Set $A_{J,k} = B_{J,k}$ and $\Delta_{J,k} = d(\{\Gamma_c(J,k,\cdot)\}_{c=1}^C)$ for $k = 0, ..., 2^J - 1$.*

Step 3: *Determine the best subspace* $A_{j,k}$ *for* $j = J-1, ..., 0,\ k = 0, ..., 2^j - 1$ *by the following rule:*

Set $\Delta_{j,k} = d(\{\Gamma_c(J,k,\cdot)\}_{c=1}^C)$.

If $\Delta_{j,k} \geq \Delta_{j+1,2k} + \Delta_{j+1,2k+1}$,

then $A_{j,k} = B_{j,k}$,

else $A_{j,k} = A_{j+1,2k} \oplus A_{j+1,2k+1}$ *and set* $\Delta_{j,k} = \Delta_{j+1,2k} + \Delta_{j+1,2k+1}$.

Step 4: *Order the basis functions by their power of discrimination (explained below).*

Step 5: *Use* $k \leq n$ *most discriminating basis functions for constructing classifiers.*

We note that Step 3 is fast, $O(n)$, since d is additive. In Step 4, we evaluate the power of discrimination of a single basis function $\mathbf{w}_{j,k,l}$ by $d(\{\Gamma_c(j,k,l)\}_{c=1}^C)$. That the basis obtained by the LDB-algorithm indeed has the desired property, is stated in Proposition 1.1, for which (simple) proof we refer to [**4**].

PROPOSITION 1.1. *The basis obtained by the LDB-algorithm maximizes the additive discriminant measure* d *on the time-frequency energy maps* $\{\Gamma_c\}_{c=1}^C$ *among all the bases in the dictionary* $\mathcal{D}$.

CHAPTER 2

Factorization of the Discrete Wavelet Transform in Dimension 1 Using Elements in $SO_2(\mathbf{R}), SL_m(\mathbf{R})$

1. Introduction

1.1. Notation. If T is a linear transformation, then T^t, T^{-t} will be the transpose of T, T^{-1}, respectively. If $\mathbf{x} \in \mathbf{R}^n$, then $S(k)\mathbf{x}(j) = \mathbf{x}(j-k)$ and $\sigma\mathbf{x}(j) = \mathbf{x}(-j)$. If $a : \mathbf{Z}^n \to \mathbf{R}^n$, $b : \mathbf{Z}^n \to \mathbf{R}^n$, then $a * b(k) = \sum_{j\in\mathbf{Z}} a(j)b(k-j)$. For convenience we will occasionally denote Daubechies' shortest filters of length L, corresponding to an orthonormal compactly supported wavelet possessing a number of $L/2$ vanishing moments, by H_L^d for lowpass and G_L^d for highpass. We will denote the operation of "convolution followed by downsampling by 2" by $*_2$. Using similar notation, in the biorthogonal case we will denote the analysis lowpass- and highpass filters by H_L^b, $G_{\tilde{L}}^b$ respectively, and the synthesis lowpass and highpass filters by $\tilde{H}_{\tilde{L}}^b$, $\tilde{G}_L^b$, where L and $\tilde{L}$ are the lengths of the lowpass analysis- and lowpass synthesis filter, respectively.

1.2. The idea. We can present the main idea in our construction in just a few words. Consider a conjugate pair of FIR analysis filters H, G. The filtering of a vector in $\mathbf{R}^n$, n even, into lowpass and highpass coefficients using this pair of filters for the operation $*_2$, can be arranged as a mapping induced by a linear operator $\Theta_n(H,G) : \mathbf{R}^n \mapsto \mathbf{R}^n$ defined by

$$\Theta_n(H,G) : \mathbf{x} \in \mathbf{R}^n \mapsto \left(c_0^1, d_0^1, c_1^1, d_1^1, \cdots, c_{n/2-1}^1, d_{n/2-1}^1\right). \tag{2.1}$$

Given a vector $\mathbf{x} \in \mathbf{R}^n$, we consider the following linear operators from $\mathbf{R}^n$ to $\mathbf{R}^n$, defined by

$$F_2(\alpha) : \mathbf{x} \to \left\{ M_2(\alpha) \begin{pmatrix} x_{2j} \\ x_{2j+1} \end{pmatrix} \right\}_{j=0}^{n/2-1},$$
$$SO_2(\mathbf{R}) \ni M_2(\alpha) = \frac{1}{\sqrt{1+\alpha^2}} \begin{pmatrix} 1 & -\alpha \\ \alpha & 1 \end{pmatrix}, \tag{2.2}$$

$$\hat{F}_m(\alpha) : \mathbf{x} \to \left\{ \hat{M}_m(\alpha) \begin{pmatrix} x_{mj} \\ x_{mj+1} \\ \vdots \\ x_{mj+m-1} \end{pmatrix} \right\}_{j=0}^{n/m-1}, m \geq 2,$$

$$SL_m(\mathbf{R}) \ni \hat{M}_m(\alpha)(i,j) = \begin{cases} 1 & \text{if } i = j \\ -\alpha & \text{if } i = 1, j = m \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

$$\tilde{F}_3(\alpha) : \mathbf{x} \to \left\{ \tilde{M}_3(\alpha) \begin{pmatrix} x_{2j} \\ x_{2j+1} \\ x_{2j+2} \end{pmatrix} \right\}_{j=0}^{n/2-1},$$

$$SL_3(\mathbf{R}) \ni \tilde{M}_3(\alpha) = \begin{pmatrix} 1 & -\alpha & 0 \\ 0 & 1 & 0 \\ 0 & -\alpha & 1 \end{pmatrix}, \tag{2.4}$$

$$\bar{F}_2(\alpha,\beta) : \mathbf{x} \to \left\{ \bar{M}_2(\alpha,\beta) \begin{pmatrix} x_{2j} \\ x_{2j+1} \end{pmatrix} \right\}_{j=0}^{n/2-1},$$

$$SL_2(\mathbf{R}) \ni \bar{M}_2(\alpha) = \frac{1}{1+\alpha\beta} \begin{pmatrix} 1 & -\alpha \\ \beta & 1 \end{pmatrix}, \tag{2.5}$$

Remark. In case the number $n/m, m > 2$ is non-integer, just extend $\mathbf{x} \in \mathbf{R}^n$ by periodizing or mirroring or by zeros.

We will show how the operator $\Theta_n(H,G)$ can be factored into:

- A composition of operators $F_2(\alpha_j)$ and $S(k_j)$, j in some finite index interval, in the case where H, G is a conjugate orthogonal pair of filters.
- A composition of operators $\hat{F}_m^{-1}(\alpha_j)$, $\hat{F}_m^{-t}(\alpha_j)$ and $S(k_j)$, with m, j in some finite index intervals, in the case where H, G is a pair of biorthogonal conjugate analysis filters of odd-odd lengths or odd-even/even-odd lengths.
- A composition of operators $\tilde{F}_3^{-1}(\alpha_j)$, $\hat{F}_3^{-t}(\alpha_j)$ and $S(k_j)$, with j in some finite index interval, in the case where H, G is a pair of biorthogonal conjugate *symmetric* analysis filters of odd-odd lenghts.
- A composition of operators $\hat{F}_m^{-1}(\alpha_j)$, $\hat{F}_m^{-t}(\alpha_j)$, $\bar{F}_2^{-1}(\alpha_j)$, $\bar{F}_2^{-t}(\alpha_j)$ and $S(k_j)$, with m, j in some finite index intervals, in the case where H, G is a conjugate pair of biorthogonal filters of even-even lengths.

We will start considering Daubechies/Coiflet filters, and then extend our results to biorthogonal filters.

2. Factorization of the orthonormal wavelet transform.

Let H_L^d, G_L^d be an orthonormal pair of FIR filters. A linear orthogonal map on $\mathbf{R}^2$ is some matrix $M(\alpha)$, $\alpha \in \mathbf{R}$ in the one-parameter family $SO_2(\mathbf{R})$. If we can represent $\Theta_n(H_L^d, G_L^d)$ by some composition of the orthogonal maps $F_2(\alpha_j)$ as defined in (2.2), we have an alternative way of computing the discrete wavelet transform in the orthonormal case.

Defining the normalization factor R_m by

$$R_m = \prod_{j=1}^{m} (1 + \alpha_j^2)^{-\frac{1}{2}}, \tag{2.6}$$

then because of the linearity of the maps $F(\alpha_j)$, we can avoid repeated multiplication by square-root factors, instead normalizing in the last step by scalar-multiplication by some R_m.

From the orthogonality relation (1.5) we see that the operation of lowpass filtering when applied to the lowpass filter itself, reduces the filter to 1 point. Obviously, we have to construct a stepwise procedure to reduce the length of the lowpass filter, using some set of matrices $M_2(\alpha_j)$. We define the first step to be the operation

$$H_L^d \mapsto F_2(\alpha_1) H_L^d, \tag{2.7}$$

where we claim

$$H_L^d(0) \mapsto 0, \text{ which implies } \alpha_1 = \frac{H_L^d(0)}{H_L^d(1)}. \tag{2.8}$$

Then by (1.5) we have

$$\begin{aligned} 0 \quad &= \quad H_L^d(0) \cdot H_L^d(L-2) + H_L^d(1) \cdot H_L^d(L-1) \\ &\Longrightarrow \quad \frac{H_L^d(0)}{H_L^d(1)} = -\frac{H_L^d(L-1)}{H_L^d(L-2)} \\ &\Longrightarrow \quad H_L^d(L-1) \mapsto 0 \text{ under } F_2(\alpha_1). \end{aligned} \tag{2.9}$$

We see that the number of coefficients in the filter $F_2(\alpha_1) H_L^d$ is 2 less than in H_L^d. It is now clear how to proceed: To simplify notation we write

$$W_2(\alpha_k) \equiv F_2(\alpha_k) S(-(k-1)). \tag{2.10}$$

$W_2(\cdot)$ is an orthogonal operator. Then we define the k'th step in our procedure, $1 \le k \le L/2$ as the operation

$$W_2(\alpha_{k-1})\cdots W_2(\alpha_1)H_L^d \mapsto W_2(\alpha_k)W_2(\alpha_{k-1})\cdots W_2(\alpha_1)H_L^d. \quad (2.11)$$

where we claim $W_2(\alpha_k)\circ\cdots\circ W_2(\alpha_1)H_L^d(0)\mapsto 0$.

Because every $W_2(\alpha_j)$ is a linear orthogonal operator, the orthogonality relation (1.5) is preserved in each step. Thus each step but the last will reduce the length of the filter at the previous step by 2, the last step reducing a filter of length 2 to a filter of length 1. Our construction is illustrated in Figure 1 below for the filter H_6^d. Each cross in Figure 1 represents the mapping induced by $M_2(\alpha)$ on the lower pair of points resulting in the upper pair of points.

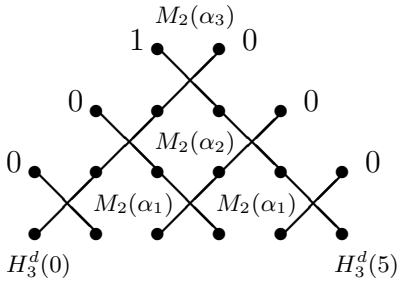

FIGURE 1. Orthogonal mapping of the lowpass filter H_6^d to 1 point.

Since the highpass filter G_L^d is the alternating flip of H_L^d (see relation (1.6)), it is easy to see that applying the operation defined in (2.11) to G_L^d will reduce it to a 1-point-filter. Because of the orthogonality of lowpass channel to highpass channel, we see that

$$H_L^d \mapsto (1,0) \Longrightarrow G_L^d \mapsto (0,1).$$

Thus we get the following result:

THEOREM 2.1. *Given a pair of orthonormal FIR filters H_L^d, G_L^d of length L, the operator $\Theta_n(H_L^d, G_L^d)$ defined in (2.1) may be decomposed as $\Theta_n(H_L^d, G_L^d) = S(1-L/2)W_2(\alpha_{L/2})W_2(\alpha_{L/2-1})\cdots W_2(\alpha_1)$, with $W_2(\alpha_j)$, $1\le j\le L/2$ defined as shown above.*

PROOF. The theorem was proved by construction. The argumentation below is just a formalization of this construction.

$$\begin{aligned}
c_k^1 &= \left(\mathbf{x} *_2 \sigma H_L^d\right)(k) = \sum_{j\in\mathbf{Z}} \mathbf{x}(j)H_L^d(j-2k) = \left\langle S(2k)H_L^d, \mathbf{x}\right\rangle \\
&= \left\langle W_2^{-1}(\alpha_1)\cdots W_2^{-1}(\alpha_{L/2})W_2(\alpha_{L/2})\cdots W_2(\alpha_1)S(2k)H_L^d, \mathbf{x}\right\rangle \\
&= \left\langle W_2(\alpha_{L/2})\cdots W_2(\alpha_1)S(2k)H_L^d, W_2^{-t}(\alpha_{L/2})\cdots W_2^{-t}(\alpha_1)\mathbf{x}\right\rangle \\
&= \left\langle \delta(j, 2k-(L/2-1)), W_2(\alpha_{L/2})\cdots W_2(\alpha_1)\mathbf{x}\right\rangle \\
&= W_2(\alpha_{L/2})\cdots W_2(\alpha_1)\mathbf{x}(2k-(L/2-1)). \qquad (2.12)
\end{aligned}$$

$$\begin{aligned} d_k^1 &= \left(\mathbf{x} *_2 \sigma G_L^d\right)(k) = \sum_{j\in\mathbf{Z}} \mathbf{x}(j)G(j-2k) = \left\langle S(2k)G_L^d, \mathbf{x}\right\rangle \\ &= \left\langle W_2^{-1}(\alpha_1)\cdots W_2^{-1}(\alpha_{L/2})W_2(\alpha_{L/2})\cdots W_2(\alpha_1)S(2k)G_L^d, \mathbf{x}\right\rangle \\ &= \left\langle W_2(\alpha_{L/2})\cdots W_2(\alpha_1)S(2k)G_L^d, W_2^{-t}(\alpha_{L/2})\cdots W_2^{-t}(\alpha_1)\mathbf{x}\right\rangle \\ &= \left\langle \delta(j, 2k+1-(L/2-1)), W_2(\alpha_{L/2})\cdots W_2(\alpha_1)\mathbf{x}\right\rangle \\ &= W_2(\alpha_{L/2})\cdots W_2(\alpha_1)\mathbf{x}(2k+1-(L/2-1)). \end{aligned} \tag{2.13}$$

□

A schematic picture of the algorithm that results from the factorization of $\Theta_n(H_L^d, G_L^d)$ is shown in Figure 2 below.

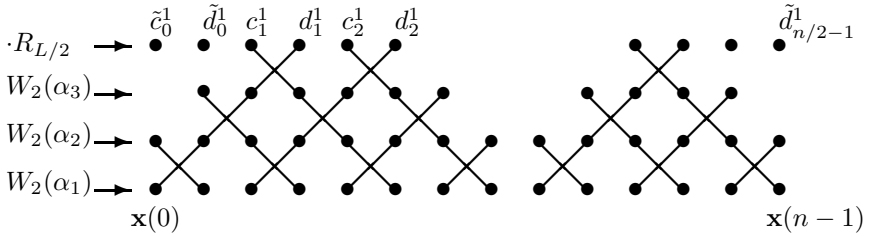

FIGURE 2. Filtering-diagram for H_6^d, G_6^d.

The "tilde" c's and d's at the edges of the filtering-diagram above can be obtained by mirroring $\mathbf{x}$ about its endpoints or periodizing. Counting operations, we see that we have to do 2 multiplications and 2 additions pr. pair of points for each $W_2(\alpha)$. Writing

$$\Pi_n(H_L^d, G_L^d) \equiv \prod_{j=L/2}^{1} W_2(\alpha_j), \tag{2.14}$$

then by construction $S(1-L/2)\Pi_n = \Theta_n$, but it has a more efficient implementation as shown in Table 1. [1]

Operation	Π_n	Θ_n
♯ mult's	$(L/2+1)\cdot n$	$L\cdot n$
♯ add's	$L/2\cdot n$	$(L-1)\cdot n$

TABLE 1. The operation cost of filtering a set of n points using the pair of filters H_L^d, G_L^d by 2 different techniques.

[1] The addition of 1 to $L/2$ results from normalization by scalar-multiplication by $R_{L/2}$.

We note that by Theorem 2.1 the inverse wavelet transform has a decomposition given by reversing the order of the operators $W_2(\alpha_j)$ and replacing each by its inverse.

3. Factorization of the biorthogonal wavelet transform.

Let $H_L^b, G_{\tilde{L}}^b$ and $\tilde{H}_{\tilde{L}}^b, \tilde{G}_L^b$ be pairs of biorthogonal FIR analysis filters and synthesis filters, respectively. We make no restrictions on them, except that they satisfy the relations given in (1.13).

We want to find a factoriztion of $\Theta_n(H_L^b, G_{\tilde{L}}^b)$ like we did in the case where the filters were orthogonal. This may seem slightly more complicated because of two major differences:

- The filter lengths L, $\tilde{L}$ of the lowpass and highpass filter need not be even numbers, moreover they will in general not be equal: $\tilde{L} - L$ may be any number.
- The highpass filter $G_{\tilde{L}}^b$ is *not* the alternating flip of the lowpass filter H_L^b, it is the alternating flip of the synthesis lowpass filter $\tilde{H}_{\tilde{L}}^b$.

But as we will see below, these differences are not crucial to the main idea in the method developed in the orthogonal case, and the relations (1.12), (1.13) will provide us with all that we need to make a modified version of the method work in this case. We simply have to remember that when dealing with the biorthogonal wavelet transform, we have to work with two "Multiresolution Analysises" (MRA's) which are *duals* of each other: One for analysis, the other for synthesis. Thus, to each linear operation/mapping we choose to perform on the analysis MRA, there corresponds a linear *dual* mapping on the synthesis MRA and vice versa. Having this in mind, we proceed to solve our problem.

Now that we have the procedure from the orthonormal case to guide us, we begin with the definition of a linear map that reduces the length of the analysis lowpass filter H_L^b. Since this will uniquely define some dual map on the synthesis lowpass filter, we see that we could equally well define our mapping on $\tilde{H}_{\tilde{L}}^b$. It will be more practical to start with the shortest filter. We assume $\tilde{L} \geq L$. Since the filter H_L^b is not double-shift orthonormal to itself, $F_2(\alpha)$ will only reduce this filter in one end for some α. Therefore, we may as well replace $F_2(\alpha)$ by $\hat{F}_2(\alpha)$ to save useless operations. We define the first step analogous to (2.7)

$$H_L^b \mapsto \hat{F}_2(\alpha_1) H_L^b, \tag{2.15}$$

where we claim

$$H_L^b(0) \mapsto 0, \text{ which implies } \alpha_1 = \frac{H_L^b(0)}{H_L^b(1)}. \tag{2.16}$$

From the "double shift biorthogonality relation" (1.12), we get

$$\begin{aligned}\delta_{0,k} &= \left\langle H_L^b, S(2k)\tilde{H}_{\tilde{L}}^b \right\rangle \\ &= \left\langle \hat{F}_2^{-1}(\alpha_1)\hat{F}_2(\alpha_1)H_L^b, S(2k)\tilde{H}_{\tilde{L}}^b \right\rangle \\ &= \left\langle \hat{F}_2(\alpha_1)H_L^b, \hat{F}_2^{-t}(\alpha_1)S(2k)\tilde{H}_{\tilde{L}}^b \right\rangle . \end{aligned} \tag{2.17}$$

Using relation (1.12) once again, we observe that

$$\begin{aligned} 0 &= H_L^b(0)\tilde{H}_{\tilde{L}}^b(\tilde{L}-2) + H_L^b(1)\tilde{H}_{\tilde{L}}^b(\tilde{L}-1) \\ \Longrightarrow\ & \frac{\tilde{H}_{\tilde{L}}^b(\tilde{L}-1)}{\tilde{H}_{\tilde{L}}^b(\tilde{L}-2)} = -\frac{H_L^b(0)}{H_L^b(1)} = -\alpha_1 \\ \Longrightarrow\ & \hat{F}_2^{-t}(\alpha_1) : \tilde{H}_{\tilde{L}}^b(\tilde{L}-1) \to 0. \end{aligned} \tag{2.18}$$

Figure 3 illustrates this fact for the case $L = 3$, $\tilde{L} = 5$.

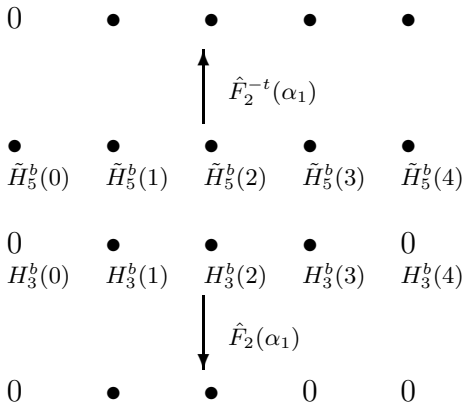

FIGURE 3. Illustration of $\hat{F}_2(\alpha_1)H_3^b$ and $\hat{F}_2^{-t}(\alpha_1)\tilde{H}_5^b$.

Then, using relation (1.13) and the duality of $\hat{F}_2(\alpha_1)$ to $\hat{F}_2^{-t}(\alpha_1)$ we get

$$\hat{F}_2(\alpha_1)G_{\tilde{L}}^b(0) = (-1)^1\hat{F}_2^{-t}(\alpha_1)\tilde{H}_{\tilde{L}}^b(\tilde{L}-1) = 0. \tag{2.19}$$

Now we have to be careful in defining the next steps. We look for a sequence of linear non-singular operators of type $\hat{F}_2(\cdot), \hat{F}_2^t(\cdot)$, the composition of which reduces the pair H_L^b, G_L^b to a pair of 1-point filters. By the relations (2.17), (2.18), (2.19), we see that this goal is reached once we have a sequence of pairs of non-singular dual linear maps that reduces the pair of filters $H_L^b, \tilde{H}_{\tilde{L}}^b$ to a pair of *biorthogonal* 1-point lowpass filters. We will show that such a sequence exists by construction.

Using the operators $\hat{F}_2(\alpha)$, each step in this construction will reduce the filter H_L^b in one end only. We have to decide which end to reduce at step k such that at every step j, $1 \leq k \leq j < max(\tilde{L}, L)$,

$$supp(\tilde{H}^b_{\tilde{L}-j}) \cap supp(H^b_{L(j)}) \neq \emptyset, \tag{2.20}$$

and such that after the last step we are left with two dual 1-point filters.

Since dual filters always have common center, we see that condition (2.20) forces a symmetric reduction of the filter lengths around the common center. As is easily checked out, this is achieved by alternatingly making use of suitable operators $\hat{F}_m(\alpha_k), \hat{F}^t_m(\alpha_{k+1}),\ m \geq 2,\ 1 \leq k < max(\tilde{L}, L) - 1$, together with suitable shifts. By the definition of the highpass filter $G^b_{\tilde{L}}$ given in (1.13), we see that the centers of the lowpass and highpass filters do not share support, the highpass center is delayed 1 time step with regard to the lowpass center. Thus, the symmetric reduction of the highpass filter is delayed 1 time step with regard to the symmetric reduction of the lowpass filter.

We note one possible problem: If at some step j, $1 \leq j < max(\tilde{L}, L)$ the filter $\tilde{H}^b_{\tilde{L}-j}$ has internal zero coefficients, we will not be able to map the outer (with regard to the center of the filter) neighbour coefficient to zero using any $\hat{F}_2(\alpha)$ or $\hat{F}^t_2(\alpha)$. But it is easy to see that by replacing $\hat{F}_2(\alpha)$ by $\hat{F}_3(\alpha)$ or $\hat{F}^t_2(\alpha)$ by $\hat{F}^t_3(\alpha)$, as defined in (2.3), the desired zero-mapping of the neighbour coefficient can be achieved in both $\tilde{H}^b_{\tilde{L}-j}$ and $H^b_{L(j)}$. In general, if there are m consecutive internal zero coefficients, one should use $\hat{F}_{m+2}(\alpha), \hat{F}^t_{m+2}(\alpha)$.

There is really not much left to prove, except to clear up some details. We separate the biorthogonal filters into 3 classes, and give the form of our factorization of the wavelet transform in each of these classes.

3.1. The general odd-odd case. Here $L \neq \tilde{L}$ are both odd numbers. To simplify notation we define

$$J = \frac{L + \tilde{L}}{2},$$
$$\hat{W}_{m(k)}(\alpha_k) \equiv \begin{cases} \hat{F}_{m(k)}(\alpha_k) S(-\lfloor \frac{k}{2} \rfloor) & \text{if } k, 1 \leq k \leq J, \text{ is odd.} \\ \hat{F}^t_{m(k)}(\alpha_k) S(-\frac{k}{2}) & \text{if } k, 1 \leq k \leq J, \text{ is even.} \end{cases} \tag{2.21}$$

Then we can state the following result:

THEOREM 2.2. *Given a pair of biorthogonal FIR filters $H^b_L, G^b_{\tilde{L}}$ of odd lengths $L, \tilde{L}$, there exists a sequence of integers $\{m(j)\}_{j=1}^J \subset \mathbf{N} -$*

$\{1\}$ *such that the operator* $\Theta_n(H_L^b, G_{\tilde{L}}^b)$ *defined in (2.1) may be decomposed as* $\Theta_n(H_L^b, G_{\tilde{L}}^b) = S(-\lfloor J/2 \rfloor)\hat{W}_{m(J)}^{-t}(\alpha_J)\hat{W}_{m(J-1)}^{-t}(\alpha_{J-1})\cdots\hat{W}_{m(1)}^{-t}(\alpha_1)$, *with* $\hat{W}_{m(j)}(\alpha_j)$, $1 \le j \le J$ *and* J *defined as in (2.21).*

PROOF. The theorem was proved by construction. The argumentation below is just a formalization of this construction.

$$\begin{aligned}
c_k^1 &= \left(\mathbf{x} *_2 \sigma H_L^b\right)(k) = \sum_{j\in\mathbf{Z}} \mathbf{x}(j)H(j-2k) = \left\langle S(2k)H_L^b, \mathbf{x}\right\rangle \\
&= \left\langle \hat{W}_{m(1)}^{-1}(\alpha_1)\cdots\hat{W}_{m(J)}^{-1}(\alpha_J)\hat{W}_{m(J)}(\alpha_J)\cdots\hat{W}_{m(1)}(\alpha_1)S(2k)H_L^d, \mathbf{x}\right\rangle \\
&= \left\langle \hat{W}_{m(J)}(\alpha_J)\cdots\hat{W}_{m(1)}(\alpha_1)S(2k)H_L^b, \hat{W}_{m(J)}^{-t}(\alpha_J)\cdots\hat{W}_{m(1)}^{-t}(\alpha_1)\mathbf{x}\right\rangle \\
&= \left\langle \delta(j, 2k-J/2), \hat{W}_{m(J)}^{-t}(\alpha_J)\cdots\hat{W}_{m(1)}^{-t}(\alpha_1)\mathbf{x}\right\rangle \\
&= \hat{W}_{m(J)}^{-t}(\alpha_J)\cdots\hat{W}_{m(1)}^{-t}(\alpha_1)\mathbf{x}(2k-\lfloor J/2\rfloor). \qquad (2.22)
\end{aligned}$$

$$\begin{aligned}
d_k^1 &= \left(\mathbf{x} *_2 \sigma G_{\tilde{L}}^b\right)(k) = \sum_{j\in\mathbf{Z}} \mathbf{x}(j)G_{\tilde{L}}^b(j-2k) = \left\langle S(2k)G_{\tilde{L}}^b, \mathbf{x}\right\rangle \\
&= \left\langle \hat{W}_{m(1)}^{-1}(\alpha_1)\cdots\hat{W}_{m(J)}^{-1}(\alpha_J)\hat{W}_{m(J)}(\alpha_J)\cdots\hat{W}_{m(1)}(\alpha_1)S(2k)G_{\tilde{L}}^d, \mathbf{x}\right\rangle \\
&= \left\langle \hat{W}_{m(J)}(\alpha_J)\cdots\hat{W}_{m(1)}(\alpha_1)S(2k)G_{\tilde{L}}^b, \hat{W}_{m(J)}^{-t}(\alpha_J)\cdots\hat{W}_{m(1)}^{-t}(\alpha_1)\mathbf{x}\right\rangle \\
&= \left\langle \delta(j, 2k+1-J/2), \hat{W}_{m(J)}^{-t}(\alpha_J)\cdots\hat{W}_{m(1)}^{-t}(\alpha_1)\mathbf{x}\right\rangle \\
&= \hat{W}_{m(J)}^{-t}(\alpha_J)\cdots\hat{W}_{m(1)}^{-t}(\alpha_1)\mathbf{x}(2k+1-\lfloor J/2\rfloor). \qquad (2.23)
\end{aligned}$$

It remains to prove that the orthogonal length-reduction of the filters $H_L^b, \tilde{H}_{\tilde{L}}^b$ takes $\frac{L+\tilde{L}}{2}$ steps. But this is easy to see: We have

$$H_L^b \mapsto \delta_{j,0} \Longrightarrow \tilde{H}_{\tilde{L}}^b \mapsto \tilde{H}_{\tilde{L}-L+1}^b.$$

by relation (1.12). If $\tilde{L}-L = 2(2k+1), k \ge 0$, then by considering relation (1.12) we deduce that $2k$ of the $\tilde{L}-L+1$ coefficients in $\tilde{H}_{\tilde{L}-L+1}^b$ are 0, and that these 0-coefficients are distributed symmetrically around the center-coefficient. Thus the total number of steps will be

$$L-1+2k+2 = L-1+\frac{\tilde{L}-L}{2}+1 = \frac{L+\tilde{L}}{2}.$$

By a similar argument, if $\tilde{L}-L = 4k, k \ge 1$, then the total number of steps will be $\frac{L+\tilde{L}}{2}-1$.

Thus, the number J as defined in (2.21) is at least an upper bound for the total number of steps.

□

A schematic picture of the algorithm that results from the factorization of $\Theta_n(H_L^b, G_{\tilde{L}}^b)$ in this case of odd length filters is shown in Figure 4 below. At level k in the figure, each slanting line A connected to a vertical line B means multiplication of the number in the point of origin of A by the factor α_k, followed by addition to the number in the point of origin of B, while single vertical lines just mean the identity. This is easily seen to be the pairwise mapping induced by either $\hat{F}_2^{-t}(\alpha_k)$ or $\hat{F}_2^{-1}(\alpha_k)$.

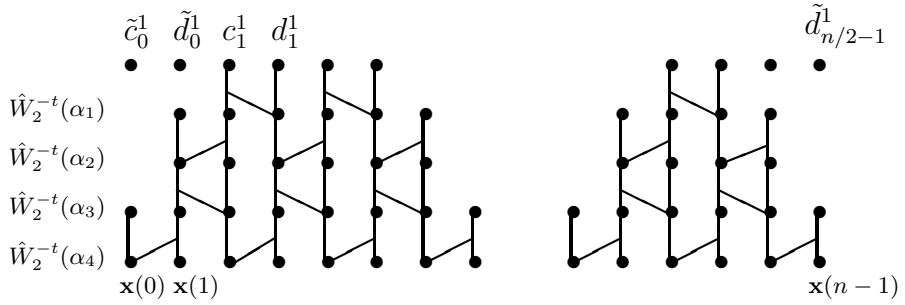

FIGURE 4. Filtering-diagram for a conjugate pair of biorthogonal filters of length 5 and 3.

The "tilde" c's and d's at the edges of the filtering-diagram above can be obtained by mirroring $\mathbf{x}$ about its endpoints or periodizing. Counting operations, we see that we have to do 1 multiplication and 1 addition pr. pair of points for each $\hat{W}_{m(j)}(\alpha_j)$. Writing

$$\Pi_n(H_L^b, G_{\tilde{L}}^b) \equiv \prod_{j=J}^{1} \hat{W}_{m(j)}^{-t}(\alpha_j), \tag{2.24}$$

then by construction $\Pi_n = S(\lfloor \frac{J}{2} \rfloor)\Theta_n$, but it has a more efficient implementation as shown in Table 2. [2]

Operation	Π_n	Θ_n
♯ mult's	$\frac{\tilde{L}+L}{4}n$	$\frac{\tilde{L}+L}{2}n$
♯ add's	$\frac{\tilde{L}+L}{4}n$	$\frac{\tilde{L}+L-2}{2}n$

TABLE 2. The operation cost of filtering a set of n points using a pair of biorthogonal odd-odd filters $H_L^b, G_{\tilde{L}}^b$ by 2 different techniques.

[2]The cost result shown in Table 2 is only an upper bound if the $H_L^b, G_{\tilde{L}}^b$ have internal zero coefficients.

We note that by Theorem 2.2 the inverse wavelet transform has a decomposition given by reversing the order of the operators $\hat{W}_2^{-t}(\alpha_j)$ and replacing each by its inverse.

3.2. The symmetric odd-odd case. There is only one crucial observation to be made here, then the results follow immediately from our work in the previous section. This is best illustrated by an example. Figure 5 shows the first 2 steps in reducing a symmetric filter of length 7 to a 1-point filter.

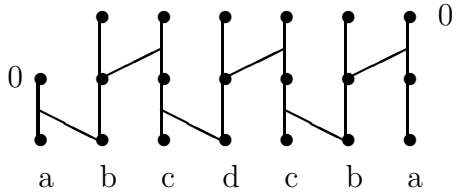

FIGURE 5. 2-step reduction of a symmetric filter of odd length using some $\hat{F}_2(\alpha_1), \hat{F}_2^t(\alpha_2)S(-1)$.

By the definition of our length reducing maps in the section above, we have $\alpha_{2k-1} = \alpha_{2k}, 1 \leq k \leq max(\tilde{L}, L)/2$, because the maps $\hat{F}_2(\alpha_{2k-1})S(-(k-1))$, $\hat{F}_2^t(\alpha_{2k})S(-k)$ preserve symmetry in pairs. Thus we see that replacing the operator $\hat{F}_2^t(\alpha_{2k})S(-k)\hat{F}_2^t(\alpha_{2k-1})S(-(k-1)), 1 \leq k \leq max(\tilde{L}, L)/2$ by the operator $\tilde{F}_3(\alpha_k)S(-(k-1)), 1 \leq k \leq max(\tilde{L}, L)/2$ as defined in (2.4), will improve efficiency: $\tilde{F}_3(\alpha_k) : H_{L-2k}^b \mapsto H_{L-2k-2}^b$.

Using similar notation to the previous section, we define

$$J = \lfloor \frac{L + \tilde{L}}{4} \rfloor,$$
$$\tilde{W}_3(\alpha_k) \equiv \tilde{F}_3(\alpha_k)S(-(k-1)). \tag{2.25}$$

Then we have the following refinement of Theorem 2.2:

COROLLARY 2.1. *Given a pair of biorthogonal symmetric FIR filters $H_L^b, G_{\tilde{L}}^b$ of odd lengths $L, \tilde{L}$, $\Theta_n(H_L^b, G_{\tilde{L}}^b)$ defined in (2.1) may be decomposed as*
$\Theta_n(H_L^b, G_{\tilde{L}}^b) = S(-\lceil \frac{J}{2} \rceil)\tilde{W}_3^{-t}(\alpha_J)\tilde{W}_3^{-t}(\alpha_{J-1}) \cdots \tilde{W}_3^{-t}(\alpha_1)$, *with* $\tilde{W}_3(\alpha_j)$, $1 \leq j \leq J$ *and* J *defined as in (2.25).*

PROOF. The only thing left to prove is that the total number of steps equals J as defined in (2.25). But this follows easily by the same argumentation as in the proof of Theorem 2.2.

□

A schematic picture of the algorithm that results from the factorization of $\Theta_n(H_L^b, G_{\tilde{L}}^b)$ in this case of symmetric odd length filters is shown in Figure 6 below with the same use of symbols as in Figure 4.

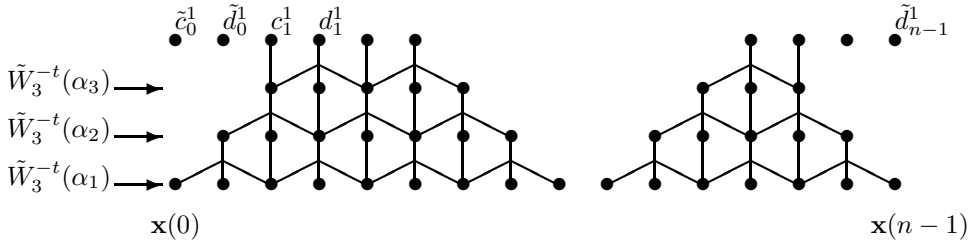

FIGURE 6. Filtering diagram for a symmetric pair of conjugate filters of length 7 and 5.

The "tilde" c's and d's at the edges of the filtering-diagram above can be obtained by mirroring $\mathbf{x}$ about its endpoints or periodizing. Counting operations, we see that we have to do 1 multiplication and 2 additions pr. pair of points for each $\tilde{W}_{m(j)}(\alpha_j)$. We write

$$\Pi_n(H_L^b, G_{\tilde{L}}^b) \equiv \prod_{j=J}^{1} \tilde{W}_{m(j)}^{-t}(\alpha_j). \tag{2.26}$$

By construction $\Pi_n = S(\lceil \frac{J}{2} \rceil - 1)\Theta_n$. Π_n has a more efficient implementation as shown in Table 3. We note that if the filters H, G have inner zero-coefficients, we simply replace $\tilde{F}_3(\alpha)$ by the composition $\hat{F}^t(\alpha)\hat{F}_m(\alpha)$, some $m \geq 3$, and it is easy to see that this does not affect the total number of additions and multiplications. [3]

Operation	Π_n	Θ_n
♯ mult's	$\frac{1}{2}\lfloor \frac{L+\tilde{L}}{4} \rfloor n$	$\frac{\tilde{L}+L}{2} n$
♯ add's	$\lfloor \frac{L+\tilde{L}}{4} \rfloor n$	$\frac{\tilde{L}+L-2}{2} n$

TABLE 3. The cost of filtering a set of n points using a pair of biorthogonal symmetric odd-odd filters $H_L^b, G_{\tilde{L}}^b$ by 2 different techniques.

3.3. The even-odd case. We assume without loss of generality that L is odd and $\tilde{L}$ is even. We observe that there is only one n such that $H_L^b(n)\tilde{H}_{\tilde{L}}^b(n) \neq 0$.

Now, define

[3] The cost result shown in Table 3 is only an upper bound if $H_L^b, G_{\tilde{L}}^b$ have internal zero coefficients.

$$J = \frac{L + \tilde{L} - 1}{2}$$

$$\hat{W}_{m(j)}(\alpha_j) \text{ as in (2.21)} \tag{2.27}$$

$$\tag{2.28}$$

Then we have the result:

COROLLARY 2.2. *Given a pair of biorthogonal FIR filters $H_L^b, G_{\tilde{L}}^b$ of odd-even lengths $L, \tilde{L}$, there exists a sequence of integers $\{m(j)\}_{j=1}^J \subset \mathbf{N} - \{1\}$ such that the operator $\Theta_n(H_L^b, G_{\tilde{L}}^b)$ defined in (2.1) may be decomposed as*
$\Theta_n(H_L^b, G_{\tilde{L}}^b) = S(1 - \lfloor \frac{J}{2} \rfloor)\hat{W}_2^{-t}(\alpha_J)\hat{W}_{m(J-1)}^{-t}(\alpha_{J-1}) \cdots \hat{W}_{m(1)}^{-t}(\alpha_1)$, *with $\hat{W}_{m(j)}(\alpha_j)$, $1 \le j \le J$ and J defined as in (2.28).*

PROOF. We only need to show that the total number of steps equals J as defined in (2.28). We have

$$H_L^b \mapsto \delta_{j,0} \Longrightarrow \tilde{H}_{\tilde{L}}^b \mapsto \tilde{H}_{\tilde{L}-L+1}^b$$

Now, setting $\tilde{L} - L = 2k + 1, k \ge 0$, we observe that $\tilde{H}_{\tilde{L}-L+1}^b$ has only $k + 1$ nonzero coefficients because of relation (1.12). This yields

$$J = L - 1 + k + 1 = L - 1 + \frac{\tilde{L} - L + 1}{2} = \frac{L + \tilde{L} - 1}{2}.$$

□

Counting operations, we see that we have to do 1 multiplication and 1 addition pr. pair of points for each $\hat{W}_{m(j)}(\alpha_j), 1 \le j \le J$.

$$\Pi_n(H_L^b, G_{\tilde{L}}^b) \equiv \prod_{j=J}^{1} \hat{W}_{m(j)}^{-t}(\alpha_j), \tag{2.29}$$

then by construction $S(1 - \lfloor \frac{J}{2} \rfloor)\Pi_n = \Theta_n$. Π_n has a more efficient implementation as shown in Table 4. [4]

3.4. The even-even case. Here, both $\tilde{L}, L$ are even numbers. There is only one important difference in this case: The last step will *not* reduce the dual filter from a 2-point filter to a 1-point filter, as illustrated in Figure 7.

This flaw is easily repaired by replacing $\hat{F}_2(\alpha_k)$ by $\bar{F}_2(\alpha_k, \beta)$ in the last step k, where β is chosen such that $\bar{F}_2^{-t}(\alpha_k, \beta)$ yields a 1-point dual filter. Using similar notation to the previous sections, we define

[4] The cost result shown in Table 4 is only an upper bound if $H_L^b, G_{\tilde{L}}^b$ have internal zero coefficients.

Operation	Π_n	Θ_n
♯ mult's	$\frac{L+\tilde{L}-1}{4}n$	$\frac{\tilde{L}+L}{2}n$
♯ add's	$\frac{L+\tilde{L}-1}{4}n$	$\frac{\tilde{L}+L-2}{2}n$

TABLE 4. The cost of filtering a set of n points using a pair of biorthogonal even-odd filters $H_L^b, G_{\tilde{L}}^b$ by 2 different techniques.

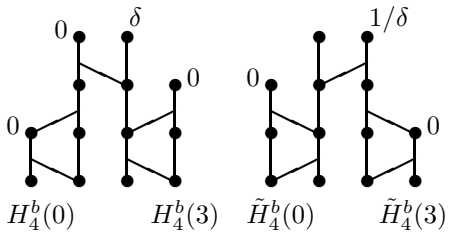

FIGURE 7. The result of reducing a pair of dual biorthogonal filters of length 4 to 1 point.

$$J = \frac{L+\tilde{L}}{2},$$
$$\hat{W}_m(\alpha_k) \equiv \begin{cases} \bar{F}_2(\alpha_J, \beta) S(-(J-1)/2) & \text{if } k = J. \\ \hat{F}_m(\alpha_k) S(-\lfloor \frac{k}{2} \rfloor) & \text{if } k, 1 \le k \le J-1, \text{ is odd.} \\ \hat{F}_m^t(\alpha_k) S(-\lfloor \frac{k-1}{2} \rfloor) & \text{if } k, 1 \le k \le J, \text{ is even.} \end{cases} \tag{2.30}$$

Then we have the following result:

COROLLARY 2.3. *Given a pair of biorthogonal FIR filters $H_L^b, G_{\tilde{L}}^b$ of even lengths $L, \tilde{L}$, there exists a sequence of integers $\{m(j)\}_{j=1}^J \subset \mathbf{N} - \{1\}$ such that the operator $\Theta_n(H_L^b, G_{\tilde{L}}^b)$ defined in (2.1) may be decomposed as*
$\Theta_n(H_L^b, G_{\tilde{L}}^b) = S(1 - \lfloor \frac{J}{2} \rfloor) \hat{W}_2^{-t}(\alpha_J) \hat{W}_{m(J-1)}^{-t}(\alpha_{J-1}) \cdots \hat{W}_{m(1)}^{-t}(\alpha_1)$, *with $\hat{W}_{m(j)}(\alpha_j)$, $1 \le j \le J$ and J defined as in (2.30).*

Counting operations, we see that we have to do 1 multiplication and 1 addition pr. pair of points for each $\hat{W}_{m(j)}(\alpha_j), 1 \le j < J$, and 2 multiplications and 2 additions pr. pair of points when implementing $\hat{W}_J(\alpha_J)$. Writing

$$\Pi_n(H_L^b, G_{\tilde{L}}^b) \equiv \prod_{j=J}^{1} \hat{W}_{m(j)}^{-t}(\alpha_j), \tag{2.31}$$

then by construction $S(1 - \lfloor \frac{J}{2} \rfloor)\Pi_n = \Theta_n$. Π_n has a more efficient implementation as shown in Table 5. [5]

Operation	Π_n	Θ_n
♯ mult's	$\frac{L+\tilde{L}+6}{4}n$	$\frac{\tilde{L}+L}{2}n$
♯ add's	$\frac{L+\tilde{L}+2}{4}n$	$\frac{\tilde{L}+L-2}{2}n$

TABLE 5. The cost of filtering a set of n points using a pair of biorthogonal even-even filters $H_L^b, G_{\tilde{L}}^b$ by 2 different techniques.

Remark. Since orthonormal FIR filters are a special case of the filters discussed in this section, we see that it is possible to reformulate the theory for orthonormal filters by exclusively using elements in $SL_2(\mathbf{R})$. Furthermore, it is possible to achieve the same operation cost result as before.

We note that in [**3**] is shown fast algorithms for computing wavelet coefficients with similar operation-counts to ours using a different approach.

[5]The cost result shown in Table 5 is only an upper bound if $H_L^b, G_{\tilde{L}}^b$ have internal zero coefficients.

CHAPTER 3

Extension of Results From Dimension 1 to Higher Dimensions.

1. Introduction.

When considering tensor wavelet bases in dimension 2 or 3, one will face the problem of filtering a set of $n \times n$ or $n \times n \times n$ points, respectively, in an effective way, where n is some power of 2. Assuming identical MRA's in all dimensions and using notation as in the previous chapter, we look for efficient implementations of $\Theta_n(H,G) \otimes \Theta_n(H,G)$, $\Theta_n(H,G) \otimes \Theta_n(H,G) \otimes \Theta_n(H,G)$. Proceeding straightforward, we can use the factorization of the wavelet transform in dimension 1 separately in each of the 2 or 3 dimensions, and thus achieve the same operation cost result as before: Reducing the operation count roughly by half. However, in both the orthonormal and biorthogonal case it is possible to obtain substantially better results.

2. Dimension 2.

2.1. The orthonormal case. Since our factored wavelet transform in dimension 1 only works with 2 points at a time, and tensor products commute, we can restrict to a square consisting of a set of 2×2 points at a time, as illustrated in Figure 1 below.

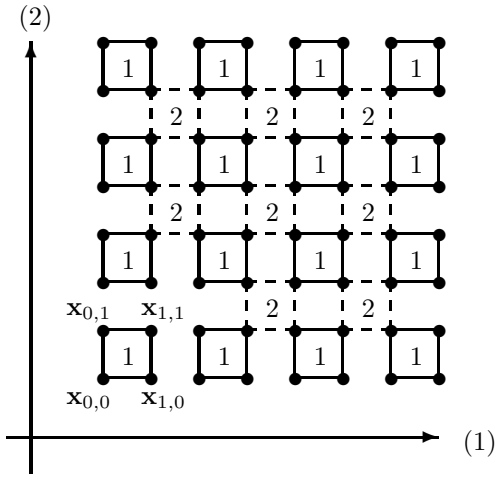

FIGURE 1. Filtering-diagram for the discrete wavelet transform in dimension 2.

The Figure 1 illustrates the division of the point-set into point-squares. Each point-square represents the map induced by restricting some $F_2(\alpha) \otimes F_2(\alpha)$ to this point set, more specificly each line connecting a pair of points represents the map induced by restricting some $F_2(\alpha)$ to this pair of points. The squares will alternatingly be in the position marked 1 and 2 as k in $F_2(\alpha_k) \otimes F_2(\alpha_k)$ varies. It will turn out favourably to use the matrix $M_2(\alpha)$ on its trigonometric form, that is

$$M_2(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}. \tag{3.1}$$

Using this form and restricting to a point-square, we write

$$F_2(\alpha) \otimes F_2(\alpha)|_{2\times 2} : \begin{Bmatrix} \mathbf{x}_{0,0} & \mathbf{x}_{1,0} \\ \mathbf{x}_{0,1} & \mathbf{x}_{1,1} \end{Bmatrix} \mapsto \begin{Bmatrix} \mathbf{y}_{0,0} & \mathbf{y}_{1,0} \\ \mathbf{y}_{0,1} & \mathbf{y}_{1,1} \end{Bmatrix}. \tag{3.2}$$

Computing on a square yields

$$\begin{pmatrix} \mathbf{y}_{0,0} \\ \mathbf{y}_{0,1} \end{pmatrix} = \begin{pmatrix} \cos^2\alpha(\mathbf{x}_{0,0} - \mathbf{x}_{1,1}) - \sin\alpha\cos\alpha(\mathbf{x}_{0,1} + \mathbf{x}_{1,0}) + \mathbf{x}_{1,1} \\ -\sin^2\alpha(\mathbf{x}_{0,1} + \mathbf{x}_{1,0}) + \sin\alpha\cos\alpha(\mathbf{x}_{0,0} - \mathbf{x}_{1,1}) + \mathbf{x}_{0,1} \end{pmatrix},$$

$$\begin{pmatrix} \mathbf{y}_{1,0} \\ \mathbf{y}_{1,1} \end{pmatrix} = \begin{pmatrix} -\sin^2\alpha(\mathbf{x}_{0,1} + \mathbf{x}_{1,0}) + \sin\alpha\cos\alpha(\mathbf{x}_{0,0} - \mathbf{x}_{1,1}) + \mathbf{x}_{1,0} \\ -\cos^2\alpha(\mathbf{x}_{0,0} - \mathbf{x}_{1,1}) + \sin\alpha\cos\alpha(\mathbf{x}_{0,1} + \mathbf{x}_{1,0}) + \mathbf{x}_{0,0} \end{pmatrix}.$$

Writing

$$d = \cos^2\alpha(\mathbf{x}_{0,0} - \mathbf{x}_{1,1}) - \sin\alpha\cos\alpha(\mathbf{x}_{0,1} + \mathbf{x}_{1,0}),$$

we get

$$\begin{aligned} \mathbf{y}_{0,0} &= d + \mathbf{x}_{1,1} \\ \mathbf{y}_{0,1} &= \cot\alpha \cdot d + \mathbf{x}_{0,1} \\ \mathbf{y}_{1,0} &= \cot\alpha \cdot d + \mathbf{x}_{1,0} \\ \mathbf{y}_{1,1} &= -d + \mathbf{x}_{0,0}. \end{aligned}$$

Combining these relations, we see that $F_2(\alpha) \otimes F_2(\alpha)|_{2\times 2}$ can be implemented by 3 multiplications and 7 additions. Using notation as in dimension 1, we write

$$\Xi_n(H_L^d, G_L^d) \equiv \prod_{j=L/2}^{1} (W_2(\alpha_j) \otimes W_2(\alpha_j))|_{2\times 2}\,, \tag{3.3}$$

Then by construction $\Xi_n = S(L/2-1)\Theta_n \otimes S(L/2-1)\Theta_n$, but Ξ_n has a more efficient implementation, as shown in Table 1. [1]

Operation	Ξ_n	$\Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n$
♯ mult's	$\frac{3}{8}Ln^2$	$(L+1)n^2$	$2Ln^2$
♯ add's	$\frac{7}{8}Ln^2$	Ln^2	$(2L-2)n^2$

TABLE 1. The operation-cost of filtering a set of $n \times n$ points using the pair of filters H_L^d, G_L^d by 3 different techniques.

2.2. The biorthogonal case. Arguing as in the last section and using the same notation as in dimension 1, we write

$$\hat{F}_2^{-t}(\alpha) \otimes \hat{F}_2^{-t}(\alpha)\Big|_{2\times 2} : \begin{Bmatrix} \mathbf{x}_{0,0} & \mathbf{x}_{1,0} \\ \mathbf{x}_{0,1} & \mathbf{x}_{1,1} \end{Bmatrix} \mapsto \begin{Bmatrix} \mathbf{y}_{0,0} & \mathbf{y}_{1,0} \\ \mathbf{x}_{0,1} & \mathbf{y}_{1,1} \end{Bmatrix} \tag{3.4}$$

$$\tilde{F}_3^{-t}(\alpha) \otimes \tilde{F}_3^{-t}(\alpha)\Big|_{3\times 3} : \begin{Bmatrix} \mathbf{x}_{0,0} & \mathbf{x}_{1,0} & \mathbf{x}_{2,0} \\ \mathbf{x}_{0,1} & \mathbf{x}_{1,1} & \mathbf{x}_{2,1} \\ \mathbf{x}_{0,2} & \mathbf{x}_{1,2} & \mathbf{x}_{2,2} \end{Bmatrix} \mapsto$$

$$\begin{Bmatrix} \mathbf{x}_{0,0} & \mathbf{y}_{1,0} & \mathbf{x}_{2,0} \\ \mathbf{y}_{0,1} & \mathbf{y}_{1,1} & \mathbf{y}_{2,1} \\ \mathbf{x}_{0,2} & \mathbf{y}_{1,2} & \mathbf{x}_{2,2} \end{Bmatrix}. \tag{3.5}$$

We see that for even-even filters and general odd-odd filters we can restrict to a set of 2×2 points at a time, while for symmetric odd-odd filters we restrict to a set of 3×3 points at a time. This "factorization" of operations allows us to implement the biorthogonal wavelet transform more efficiently in dimension 2, as illustrated in Figure 2. Each arrow-line in the figure means multiplication of the point of origin of the arrow by the number α, followed by addition of the product to the point of termination of the arrow line.

To implement $\hat{F}_2^{-t}(\alpha) \otimes \hat{F}_2^{-t}(\alpha)$ efficiently, we compute on a square of 2×2 points as shown in Figure 2. Writing

$$\begin{aligned} \mathbf{y}_{0,0} &= \mathbf{x}_{0,0} + \alpha \mathbf{x}_{0,1} \\ \mathbf{y}_{0,1} &= \mathbf{x}_{0,1} \\ \mathbf{y}_{1,1} &= \mathbf{x}_{1,1} + \alpha \mathbf{x}_{0,1} \\ \mathbf{y}_{1,1} &= \mathbf{x}_{1,0} + \alpha(\mathbf{y}_{1,1} + \mathbf{x}_{0,0}), \end{aligned}$$

[1] The addition of 1 to L in the multiply-cost for $\Pi_n \otimes \Pi_n$ results from scalar multiplication by $R_{L/2}^2$.

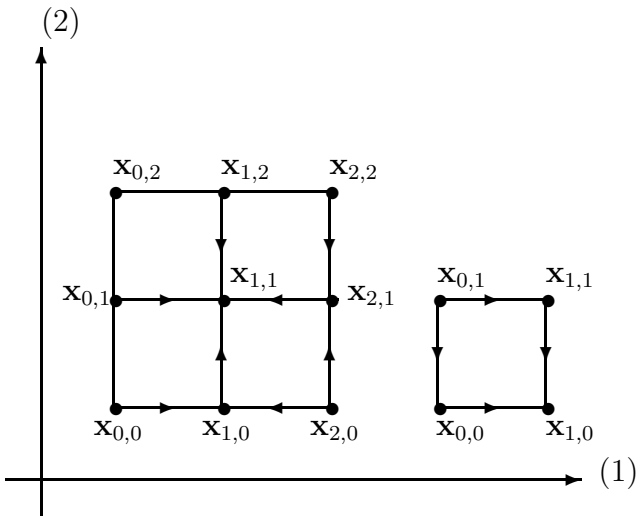

FIGURE 2. The point-squares for symmetric and non-symmetric biorthogonal filters, respectively.

we see that we can implement $\hat{F}_2^{-t}(\alpha) \otimes \hat{F}_2^{-t}(\alpha)\Big|_{2\times 2}$ by 2 multiplications and 4 additions.

Similarly, to implement $\tilde{F}_3^{-t}(\alpha) \otimes \tilde{F}_3^{-t}(\alpha)$ efficiently, we write

$$\begin{aligned}
\mathbf{y}_{0,0} &= \mathbf{x}_{0,0} \\
\mathbf{y}_{0,1} &= \mathbf{x}_{0,1} \\
\mathbf{y}_{0,2} &= \mathbf{x}_{0,2} \\
\mathbf{y}_{1,2} &= \mathbf{x}_{1,2} \\
\mathbf{y}_{2,2} &= \mathbf{x}_{2,2} \\
\mathbf{y}_{2,0} &= \mathbf{x}_{2,0} \\
\mathbf{y}_{1,0} &= \mathbf{x}_{1,0} + \alpha\mathbf{x}_{0,0} + \alpha\mathbf{x}_{2,0} \\
\mathbf{y}_{2,1} &= \mathbf{x}_{2,1} + \alpha\mathbf{x}_{2,2} + \alpha\mathbf{x}_{2,0} \\
\mathbf{y}_{1,1} &= \mathbf{x}_{1,1} + \alpha(\mathbf{y}_{1,0} + \alpha\mathbf{x}_{2,2} + \mathbf{x}_{2,1} + \mathbf{x}_{0,1} + \mathbf{x}_{1,2})
\end{aligned}$$

Thus, we see that $\tilde{F}_3^{-t}(\alpha) \otimes \tilde{F}_3^{-t}(\alpha)\Big|_{3\times 3}$ can be implemented by 2 multiplications and 8 additions, assuming that we have stored the numbers $\alpha\mathbf{x}_{2,2}, \alpha\mathbf{x}_{0,0}, \alpha\mathbf{x}_{2,2} + \mathbf{x}_{1,2}$ from previous steps in the implementation of the algorithm.

Using notation as in (2.21), in the general odd-odd case we define

$$\Xi_n(H_L^b, G_{\tilde{L}}^b) \equiv \prod_{j=J}^{1} (\hat{W}_2(\alpha_j) \otimes \hat{W}_2(\alpha_j))\Big|_{2\times 2}, \tag{3.6}$$

and thus get $\Xi_n = S(\lfloor J/2 \rfloor)\Theta_n \otimes S(\lfloor J/2 \rfloor)\Theta_n$.

In the odd-even case we define

$$\Xi_n(H_L^b, G_{\tilde{L}}^b) \equiv \prod_{j=J}^{1} (\hat{W}_2(\alpha_j) \otimes \hat{W}_2(\alpha_j))\Big|_{2\times 2}, \tag{3.7}$$

where we use the notation defined in (2.28). We get $\Xi_n = S(\lfloor \frac{J}{2} \rfloor - 1)\Theta_n \otimes S(\lfloor \frac{J}{2} \rfloor - 1)\Theta_n$.

In the even-even case we define

$$\Xi_n(H_L^b, G_{\tilde{L}}^b) \equiv \prod_{j=J}^{1} (\hat{W}_2(\alpha_j) \otimes \hat{W}_2(\alpha_j))\Big|_{2\times 2}, \tag{3.8}$$

where we use the notation defined in (2.30). We get $\Xi_n = S(\lfloor \frac{J}{2} \rfloor - 1)\Theta_n \otimes S(\lfloor \frac{J}{2} \rfloor - 1)\Theta_n$.

Finally, in the symmetric odd-odd case we define

$$\Xi_n(H_L^b, G_{\tilde{L}}^b) \equiv \prod_{j=J}^{1} (\tilde{W}_3(\alpha_j) \otimes \tilde{W}_3(\alpha_j))\Big|_{3\times 3}, \tag{3.9}$$

where we use the notation in (2.25). This yields $\Xi_n = S(\lceil \frac{J}{2} \rceil)\Theta_n \otimes S(\lceil \frac{J}{2} \rceil)\Theta_n$.

Thus, we get the operation-cost results shown in Table 2, Table 4, Table 5. [2]

Operation	Ξ_n	$\Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n$
♯ mult's	$\frac{L+\tilde{L}}{4}n^2$	$\frac{L+\tilde{L}}{2}n^2$	$(L+\tilde{L})n^2$
♯ add's	$\frac{L+\tilde{L}}{2}n^2$	$\frac{L+\tilde{L}}{2}n^2$	$(L+\tilde{L}-2)n^2$

TABLE 2. The operation cost of filtering a set of $n \times n$ points using a pair of biorthogonal odd-odd filters $H_L^b, G_{\tilde{L}}^b$ by 3 different techniques.

3. Dimension 3.

It is straightforward to see that we can group the operations of $F_2(\alpha) \otimes F_2(\alpha) \otimes F_2(\alpha)$, $\hat{F}_2(\alpha) \otimes \hat{F}_2(\alpha) \otimes \hat{F}_2(\alpha)$ into cubes of $2 \times 2 \times 2$ points, and $\tilde{F}_3(\alpha) \otimes \tilde{F}_3(\alpha) \otimes \tilde{F}_3(\alpha)$ into cubes of $3 \times 3 \times 3$ points. This is illustrated in Figure 3. To make things easy, we will use some "loose" geometrical terminology.

[2] The operation-counts shown in Table 2, Table 3, Table 4, Table 5, are only upper bounds if $H_L^b, G_{\tilde{L}}^b$ have inner zero coefficients.

Operation	Ξ_n	$\Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n$
$\sharp$ mult's	$\frac{L+\tilde{L}-1}{4}n^2$	$\frac{L+\tilde{L}-1}{2}n^2$	$(L+\tilde{L})n^2$
$\sharp$ add's	$\frac{L+\tilde{L}-1}{2}n^2$	$\frac{L+\tilde{L}-1}{2}n^2$	$(L+\tilde{L}-2)n^2$

TABLE 3. The operation cost of filtering a set of $n \times n$ points using a pair of biorthogonal odd-even filters $H^b_L, G^b_{\tilde{L}}$ by 3 different techniques.

Operation	Ξ_n	$\Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n$
$\sharp$ mult's	$\frac{L+\tilde{L}+10}{4}n^2$	$\frac{L+\tilde{L}+4}{2}n^2$	$(L+\tilde{L})n^2$
$\sharp$ add's	$\frac{L+\tilde{L}+2}{2}n^2$	$\frac{L+\tilde{L}+2}{2}n^2$	$(L+\tilde{L}-2)n^2$

TABLE 4. The operation cost of filtering a set of $n \times n$ points using a pair of biorthogonal even-even filters $H^b_L, G^b_{\tilde{L}}$ by 3 different techniques.

Operation	Ξ_n	$\Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n$
$\sharp$ mult's	$\frac{1}{2}\lfloor\frac{L+\tilde{L}}{4}\rfloor n^2$	$\lfloor\frac{L+\tilde{L}}{4}\rfloor n^2$	$(L+\tilde{L})n^2$
$\sharp$ add's	$\frac{9}{4}\lfloor\frac{L+\tilde{L}}{4}\rfloor n^2$	$2\lfloor\frac{L+\tilde{L}}{4}\rfloor n^2$	$(L+\tilde{L}-2)n^2$

TABLE 5. The operation cost of filtering a set of $n \times n$ points using a pair of biorthogonal symmetric odd-odd filters $H^b_L, G^b_{\tilde{L}}$ by 3 different techniques.

3.1. The orthonormal case. We did not succeed in finding any symmetries in the trigonometric expressions in the 3-dimensional case. The cheapest arrangement of operations will therefore be to apply $W_2(\alpha) \otimes W_2(\alpha)|_{2\times 2}$ to 2 "opposing planes" of 2×2 points in the $2\times 2\times 2$ cube, and then $W_2(\alpha)$ to the 4 "lines" that connect these 2 planes. Thus, one has to do 14 multiplications and 22 additions pr. cube, not counting normalization. We denote this "factored" wavelet transform in 3 dimensions on $n\times n\times n$ points by $\Omega_n(H^d_L, G^d_{\tilde{L}})$. Like before, we have $\Omega_n = S(L/2-1)\Theta_n \otimes S(L/2-1)\Theta_n \otimes S(L/2-1)\Theta_n$. The operation cost-result is shown in Table 6. [3]

3.2. The biorthogonal case. By rearranging the arithmetic operations, we find we have to do 6 multiplications and 12 additions pr. cube in the even-even/odd-even and non-symmetric odd-odd case. In

[3]The addition of 1 to $\frac{7}{8}L$ and $\frac{3}{2}L$ is because of normalization by $R^3_{L/2}$.

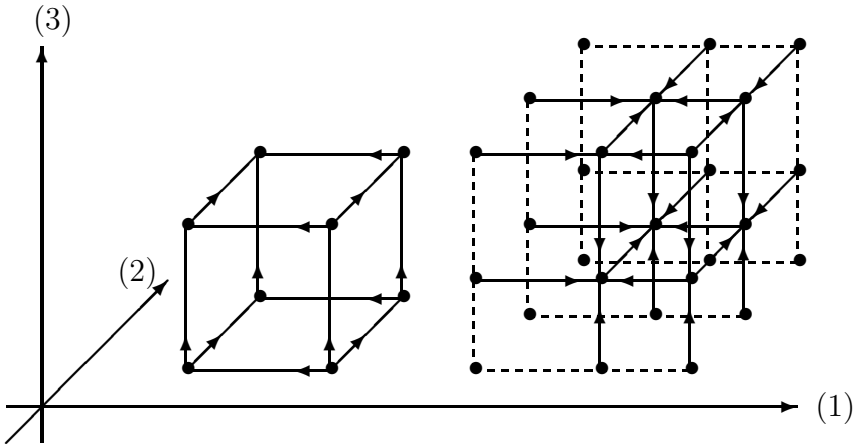

FIGURE 3. The cube for biorthogonal odd-odd/even-even filters on the left and for symmetric odd-odd filters on the right.

Operation	Ω_n	$\Pi_n \otimes \Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n \otimes \Theta_n$
♯ mult's	$(\frac{7}{8}L+1)n^3$	$(\frac{3}{2}L+1)n^3$	$3Ln^3$
♯ add's	$\frac{11}{8}Ln^3$	$\frac{3}{2}Ln^3$	$3(L-1)n^3$

TABLE 6. The cost of filtering a set of $n \times n \times n$ points using the pair of filters H_L^d, G_L^d, by 3 different techniques.

the symmetric odd-odd case we found an operation arrangement yielding 6 multiplications and 24 additions pr. cube. All these operation counts are easily verified by considering Figure 3. Denoting in each case the factored biorthogonal discrete wavelet transform on $n \times n \times n$ points by $\Omega_n(H_L^b, G_{\tilde{L}}^b)$, we get that up to shifts, Ω_n equals $\Theta_n \otimes \Theta_n \otimes \Theta_n$. The cost-results are shown in Table 7, Table 8, Table 9 and Table 10. [4]

Operation	Ω_n	$\Pi_n \otimes \Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n \otimes \Theta_n$
♯ mult's	$\frac{3}{8}(L+\tilde{L})n^3$	$\frac{3}{4}(L+\tilde{L})n^3$	$\frac{3}{2}(\tilde{L}+L)n^3$
♯ add's	$\frac{3}{4}(L+\tilde{L})n^3$	$\frac{3}{4}(L+\tilde{L})n^3$	$\frac{3}{2}(L+\tilde{L}-2)n^3$

TABLE 7. The cost of filtering a set of $n \times n \times n$ points using a pair of odd-odd filters $H_L^b, G_{\tilde{L}}^b$, by 3 different techniques.

[4] If $H_L^b, G_{\tilde{L}}^b$ has inner zero-coefficients, the operation counts in Table 7, Table 8, Table 9 and Table 10 are only upper bounds.

Operation	Ω_n	$\Pi_n \otimes \Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n \otimes \Theta_n$
$\sharp$ mult's	$\frac{3}{8}(L+\tilde{L}-1)n^3$	$\frac{3}{4}(L+\tilde{L}-1)n^3$	$\frac{3}{2}(L+\tilde{L})n^3$
$\sharp$ add's	$\frac{3}{4}(L+\tilde{L}-1)n^3$	$\frac{3}{4}(L+\tilde{L}-1)n^3$	$\frac{3}{2}(L+\tilde{L}-2)n^3$

TABLE 8. The cost of filtering a set of $n \times n \times n$ points using a pair of odd-even filters $H^b_L, G^b_{\tilde{L}}$, by 3 different techniques.

Operation	Ω_n	$\Pi_n \otimes \Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n \otimes \Theta_n$
$\sharp$ mult's	$\frac{3}{8}(L+\tilde{L}+\frac{14}{3})n^3$	$\frac{3}{4}(L+\tilde{L}+\frac{10}{3})n^3$	$\frac{3}{2}(L+\tilde{L})n^3$
$\sharp$ add's	$\frac{3}{4}(L+\tilde{L})n^3$	$\frac{3}{4}(L+\tilde{L}+2)n^3$	$\frac{3}{2}(L+\tilde{L}-2)n^3$

TABLE 9. The cost of filtering a set of $n \times n \times n$ points using a pair of even-even filters $H^b_L, G^b_{\tilde{L}}$, by 3 different techniques.

Operation	Ω_n	$\Pi_n \otimes \Pi_n \otimes \Pi_n$	$\Theta_n \otimes \Theta_n \otimes \Theta_n$
$\sharp$ mult's	$\frac{3}{4}\lfloor\frac{L+\tilde{L}}{4}\rfloor n^3$	$\frac{3}{2}\lfloor\frac{L+\tilde{L}}{4}\rfloor n^3$	$\frac{3}{2}(\tilde{L}+L)n^3$
$\sharp$ add's	$3\lfloor\frac{L+\tilde{L}}{4}\rfloor n^3$	$3\lfloor\frac{L+\tilde{L}}{4}\rfloor n^3$	$\frac{3}{2}(\tilde{L}+L-2)n^3$

TABLE 10. The cost of filtering a set of $n \times n \times n$ points using a the pair of symmetric odd-odd filters $H^b_L, G^b_{\tilde{L}}$, by 3 different techniques.

CHAPTER 4

The $SO_2(\mathbf{R})$-Method in Computing Filters and Edge Maps

In this chapter we exploit our 2-point factorization of the orthonormal discrete wavelet transform in the last chapter to compute filters by solving non-linear equations in several variables numerically. We also try to improve on the preservation of regularity in the discrete wavelet transform by constructing some (non-orthogonal) edge maps.

1. Computation of orthonormal filters

We start with the construction of an orthonormal filter of length 4 with a maximum number of vanishing moments compatible with its support width. That is, we want α_1, α_2 so that

$$\begin{aligned} F_2(\alpha_2)S(1)F_2(\alpha_1)\mathbf{1} &\mapsto \delta_{1,k} \\ F_2(\alpha_2)S(1)F_2(\alpha_1)\mathbf{n} &\mapsto \delta_{1,k} \end{aligned} \tag{4.1}$$

This is simple enough to do by handcalculation.

$$c_1^1 = \frac{\alpha_1\mathbf{x}(0) + \mathbf{x}(1) - \alpha_2\mathbf{x}(2) + \alpha_1\alpha_2\mathbf{x}(3)}{(1+\alpha_1^2)^{1/2}(1+\alpha_2^2)^{1/2}} \tag{4.2}$$

$$d_1^1 = \frac{\alpha_1\alpha_2\mathbf{x}(0) + \alpha_2\mathbf{x}(1) + \mathbf{x}(2) - \alpha_1\mathbf{x}(3)}{(1+\alpha_1^2)^{1/2}(1+\alpha_2^2)^{1/2}} \tag{4.3}$$

We insert the vanishing moment conditions from (4.1) and get the equation-set

$$\left[\begin{array}{l} \alpha_1\alpha_2 + \alpha_2 - \alpha_1 + 1 = 0 \\ \alpha_2 - 3\alpha_1 + 2 = 0 \end{array} \right]$$

Solving yields

$$(\alpha_1, \alpha_2) \in \{(1/\sqrt{3}, \sqrt{3}-2), (-1/\sqrt{3}, 2+\sqrt{3})\}$$

We find the lowpass filter by inserting the solutions in (4.2).

The two solution-sets are mirror-images of each other, it is the filter shown in Table 1 that corresponds to our choice of rotation-matrix $M_2(\alpha_k)$.

n	$H_4(n)$	α_n
0	0.482962913	
1	0.836516304	$1/\sqrt{3}$
2	0.224143868	$\sqrt{3}-2$
3	-0.129409523	

TABLE 1. The filter H_4 and the entries in the corresponding $M_2(\alpha_n)$.

To find orthonormal filters of length L, the above method leads to the wearying task of solving $\frac{L}{2}$ nonlinear equations in $\frac{L}{2}$ variables. However, by use of some Maple-routines we succeeded in generating H_6^d and *some* orthogonal filter H_8 corresponding to a wavelet with 4 vanishing moments.

n	$\mathbf{H}_8^d(n)$	α_n
0	.03222310057	
1	- .0126039675	- 2.556583915
2	- .09921954341	- .1434214911
3	.2978577953	.7958755204
4	.8037387510	- 2.351285662
5	.4976186668	
6	- .02963552763	
7	- .07576571472	

TABLE 2. The computed filter H_8 and the entries in the corresponding $M_2(\alpha_k)$.

Remark: The filter H_8 in Table 2 is not the one given in [**1**], but these filters are not uniquely determined by claiming a maximum number of vanishing moments only.

Now we turn to coiflets. Claiming vanishing moments on the scaling function ϕ leads to the following observation:

$$\text{If } \sum_{n=0}^{L-1} H(n)\cdot n^p = 0 \quad \text{for } \forall\ p,\ 1 \le p \le M \le \frac{L}{2} - 1,$$

where M is odd, then

$$\sum_{n=0}^{L-1} H(n)\cdot n^{M+1} = 0. \tag{4.4}$$

This means that it suffices to impose odd vanishing moments on ϕ. Counting degrees of freedom in the filtering-diagram we find that for $H_L^{coiflet}$, if the highpass-filter has M vanishing moments, then the lowpassfilter has $2(\frac{L}{2} - M)$ vanishing moments. The proof of this is given below. Maple were able to solve the nonlinear equations to give $H_6^{coiflet}$ and *some* $H_8^{coiflet}$, where we used 3 out 4 degrees of freedom

to provide the highpass filter with 3 vanishing moments, yielding 2 vanishing moments on the scaling function.

n	$H_8^{coiflet}(n)$	α_n
-3	- .07342587213	
-2	- .03314563036	
-1	.4854426593	
0	.8065436723	
1	.3100524696	2.215250436
2	- .09943689110	.1504720765
3	- .01496247550	- .7384168123
4	.03314563037	- .451416230

TABLE 3. The computed filter $H_8^{coiflet}$ and the entries in the corresponding $M_2(\alpha_n)$. The highpass filter has 3 vanishing moments, the lowpass filter has 2 vanishing moments.

Proof of (4.4):

We have

$$0 = \int x^l \phi(x) dx \ , \forall \ p, \ 1 \leq p \leq M \leq \frac{L}{2} - 1$$
$$\iff \qquad \frac{d^p}{d\xi^p} \widehat{\phi} \Big|_{\xi=0} = 0, \text{ for } \forall \ p, \ 1 \leq p \leq M \leq \frac{L}{2} - 1.$$

Fourier transforming equation (1.1) we get

$$\widehat{\phi}(\xi) = m_0(\xi/2)\widehat{\phi}(\xi/2)$$

where m_0 is the trigonometric polynomial that is the Fourier transform of the filter H_L^{coif}. Differentiation yields

$$0 = \widehat{\phi}'(0) = \frac{1}{2} m_0'(0)\widehat{\phi}(0) + \frac{1}{2} m_0(0)\widehat{\phi}'(0) = \frac{1}{2\sqrt{\pi}} m_0'(0).$$

Repeating this argument M times we get

$$\frac{d^p}{d\xi^p} m_0 \Big|_{\xi=0} = 0, \ \text{ for } \forall \ p, \ 1 \leq p \leq M,$$

thus we conclude that m_0 must be on the form

$$m_0(\xi) = 1 + \left(1 - e^{-i\xi}\right)^{M+1} P(\xi) \tag{4.5}$$

for some trigonometric polynomial P. Assuming that the coefficients $H_L^{coif}(n)$ of m_0 are real, we have

$$m_0(-\xi) = \Re\, m_0(\xi) - i \cdot \Im\, m_0(\xi) = \overline{m_0(\xi)},$$

which implies

$$|m_0(\xi)|^2 = m_0(\xi) \cdot m_0(-\xi)$$

Expanding m_0 in a Taylorseries around the origin

$$m_0(\xi) = 1 + a_1\xi + a_2\xi^2 + \cdots + a_n\xi^n + \cdots \text{ ,where } a_k = \frac{m_0^{(k)}(0)}{k!}$$

we get

$$\begin{aligned} |m_0(\xi)|^2 &= (1 + a_1\xi + a_2\xi^2 + a_3\xi^3 + \cdots)(1 - a_1\xi + a_2\xi^2 - a_3\xi^3 + \cdots) \\ &= 1 + c_{M+1}\xi^{M+1} + c_{M+2}\xi^{M+2} + \cdots + c_{M+n}\xi^{M+n} + \cdots \end{aligned}$$

for some sequence $\{c_{M+i}\}_{i=1}^{\infty}$ because of (4.5). By computing the above product of the two series, we get the relations

$$c_0 = 1 \ , \ c_{2k+1} = 0,$$

$$c_{2k} = (-1)^k a_k^2 + 2\sum_{i=0}^{k-1}(-1)^i a_{2k-i}a_i, \text{ where } a_0 = 1. \tag{4.6}$$

Moreover

$$\begin{aligned} |a_k k!| &= \left|m_0^{(k)}(0)\right| = \left|\frac{1}{\sqrt{2}}\sum_n H_L^{coif}(n)(-in)^k e^{-in\cdot 0}\right| \\ &= \frac{1}{\sqrt{2}}\sum_n H_L^{coif}(n)n^k. \end{aligned} \tag{4.7}$$

Now, claiming the $c_k = 0$ for $1 \le k \le M$ and using (4.6), (4.7), the result follows. Indeed, one has

$$\begin{aligned} &\sum_{n=0}^{L-1} H_L^{coif}(n)n^p = 0 \ \text{ for } \forall\ p,\ 1 \le p \le M \le \frac{L}{2} - 1, \\ &\text{where } M \text{ is odd,} \\ \Longrightarrow \quad &\sum_{n=0}^{L-1} H_L^{coif}(n)n^{M+1} = \sum_{n=0}^{L-1} H_L^{coif}(n)n^{M+3} = 0. \end{aligned}$$

□

2. Ways of dealing with the edge problem

We look for ways of "smoothing" the edges of a vector in $\mathbf{R}^n$. That is, we look for operators acting on the edge coefficients that to some degree possess the following properties:

- **P1**: Linearity and orthogonality.
- **P2**: In some strict sense, "nice functions" should map to "nice functions".
- **P3**: The map should ensure some vanishing moments on the $\tilde{d}$'s.

We have to make clear the meaning of **P2**: Considering the coefficients c_k at some level j, we see from

$$\begin{aligned}\int x^M \phi_{j,k}(x)dx &= 2^{j/2}\int x^M \phi(2^j x - k)dx \\ &= 2^{-j/2}\int (2^{-j}(x+k))^M \phi(x)dx \\ &= 2^{-j(M+\frac{1}{2})}\int (x+k)^M \phi(x)dx \\ &= P_M(k)\ ,\ P_M \text{ a polynomial of degree M,}\end{aligned} \tag{4.8}$$

that a polynomial maps to a polynomial of the same degree under $\Theta_n(H_L, G_L)$. Depending on the number of vanishing moments on the corresponding ψ, the d_k's will be zero.

2.1. The method of periodizing. This method simply periodizes the vector around its edges. This yields orthogonality of the transform, but only preserves regularity up to constant functions, and we get only 1 vanishing moment on the $\tilde{d}_k$'s.

2.2. The method of mirroring. One way to overcome the edge problem is to mirror the input vector about each of its endpoints and then apply the filter to the mirrored points.

The method of mirroring does not lead to an orthogonal transform, but it preserves regularity up to continuity, and yields 1 vanishing moment on the $\tilde{d}_k$'s.

2.3. The method of edge matrices. Another approach is to try to construct a matrix map acting directly on the edgepoints such that the properties **P1**, **P2** and **P3** are satisfied to some extent. We will do this construction by claiming some vanishing moments on the $\tilde{d}_k$'s and claiming some degree of preservation of regularity of the transform at edges by claiming monomials up to some degree M mapping continuously to polynomials of degree M. That is, we compute the polynomials P_j given in (4.8) for $0 \leq j \leq M$, and impose $x^j \to P_j$,$0 \leq j \leq k(L) < M$ at edges, L is the length of lowpass filter.

For a orthonormal FIR filter H_L we will construct two $(\frac{L}{2}-1)\times(\frac{L}{2}-1)$ matrices $\mathbf{E}_{H_L,l}$, $\mathbf{E}_{H_L,r}$, each acting on the ordered set $\mathbf{e}_l$, $\mathbf{e}_r$ of $\frac{L}{2}-1$ edgepoints on the lefthand and righthandside, respectively, to give the "tilde" coefficients. For example

$$\mathbf{E}_{H_{10},l} : \mathbf{e}_l \mapsto (\tilde{c}_0,\ \tilde{d}_0,\ \tilde{c}_1,\ \tilde{d}_1)$$

$$\mathbf{E}_{H_{10},r} : \mathbf{e}_r \mapsto (\tilde{c}_{n/2-2},\ \tilde{d}_{n/2-2},\ \tilde{c}_{n/2-1},\ \tilde{d}_{n/2-1})$$

This is illustrated in Figure 1.

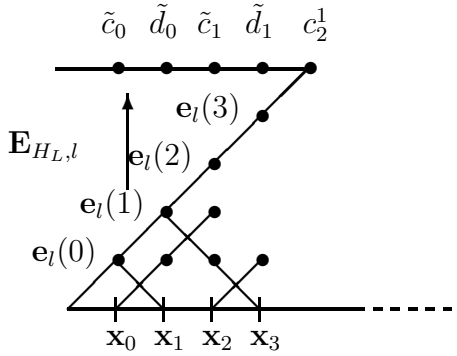

FIGURE 1.

As for fulfilling the above listed properties, this edge map is of course linear, but not orthogonal. We have sacrificed orthogonality to be able to achieve **P2** and **P3**. Some edge-matrices are shown in the appendix.

2.4. Comparing the methods. The profits of the edge matrices are obvious:

- **Good**: Preservation of regularity to a higher degree than by simply mirroring at edges.

The drawbacks are equally obvious:

- **Bad**: This edgemap is very far from orthogonal in the sense that the operator norm of the map turns out to be large, and it seems to grow with increasing degree of preservation of regularity as can be seen in Table 4 and Table 5.

To be able to compare the methods of mirroring and edge matrices in this non-orthogonality respect, we proceed as follows:

Given the vector $\mathbf{x} \in \mathbf{R}^n$ and some orthogonal FIR filter H_L, we compute the $\tilde{c}_k$'s and the $\tilde{d}_k$'s, getting

$$\tilde{c}_k = \sum_{i=0}^{K(k)} \gamma(i)\mathbf{x}(i) \tag{4.9}$$

when we use edge matrices, and

$$\tilde{c}_k = \sum_{i=0}^{J(k)} \epsilon(i)\mathbf{x}(i) \tag{4.10}$$

when we mirror $\mathbf{x}$. The expressions for the $\tilde{d}_k$'s are similar. We use Cauchy-Schwartz on (4.9) and (4.10). On the left edge we set

$$\begin{aligned} Q_l(c_k) &= \left(\sum_{i=0}^{K(k)} |\gamma(i)|^2 \right)^{1/2} \\ P_l(c_k) &= \left(\sum_{i=0}^{J(k)} |\epsilon(i)|^2 \right)^{1/2} . \end{aligned}$$

On the right edge, the bounds Q_r, P_r are defined similarly. Now, we may compare the two methods in the non-orthogonality respect. Results corresponding to the edge matrices shown in the appendix are given in Table 4 and Table 5.

l^2 bounds on the $\lvert\tilde{c}_k\rvert$, $\lvert\tilde{d}_k\rvert$.	L in H^d_L.			
	6	8	10	12
$P_l(\tilde{c}_0)$	.852	.935	1.05	1.04
$Q_l(\tilde{c}_0)$	9.50	21.8	310	757
$P_l(\tilde{d}_0)$	.989	.907	.973	.963
$Q_l(\tilde{d}_0)$	1.39	2.14	11.0	2.02
$P_l(\tilde{c}_1)$			1.04	1.12
$Q_l(\tilde{c}_1)$			48.0	103
$P_l(\tilde{d}_1)$		.994	.999	.996
$Q_l(\tilde{d}_1)$		2.59	1.58	7.98
$P_l(\tilde{d}_2)$				1.00
$Q_l(\tilde{d}_2)$				1.10
$P_r(\tilde{c}_0)$	.852	1.01	1.05	1.01
$Q_r(\tilde{c}_0)$	1.01	1.03	1.01	1.03
$P_r(\tilde{d}_0)$	.989	.994	.973	.963
$Q_r(\tilde{d}_0)$	.512	.402	.973	3.13
$P_r(\tilde{c}_1)$		.935	1.04	1.04
$Q_r(\tilde{c}_1)$		1.00	1.00	1.00
$P_r(\tilde{d}_1)$			.999	.996
$Q_r(\tilde{d}_1)$			2.58	5.03
$P_r(\tilde{c}_2)$				1.12
$Q_r(\tilde{c}_2)$				1.00

TABLE 4. The l^2 bounds for the edge operators $\mathbf{E}_{H^d_L,\cdot}$ using Daubechies shortest filters.

l^2 bounds on the $\lvert\tilde{c}_k\rvert$, $\lvert\tilde{d}_k\rvert$.	L in H_L^{coif}.		
	6	8	12
$P_l(\tilde{c}_0)$	1.13	.949	1.16
$Q_l(\tilde{c}_0)$	4.75	2.43	25.0
$P_l(\tilde{d}_0)$	.930	1.15	.909
$Q_l(\tilde{d}_0)$	1.97	10.4	2.31
$P_l(\tilde{c}_1)$			.997
$Q_l(\tilde{c}_1)$			2.50
$P_l(\tilde{d}_1)$		1.02	.999
$Q_l(\tilde{d}_1)$		1.54	2.35
$P_l(\tilde{d}_2)$			1.00
$Q_l(\tilde{d}_2)$			2.31
$P_r(\tilde{c}_0)$	1.13	.907	.997
$Q_r(\tilde{c}_0)$	1.02	1.97	1.45
$P_r(\tilde{d}_0)$	.930	1.02	.999
$Q_r(\tilde{d}_0)$	.705	.908	3.33
$P_r(\tilde{c}_1)$		.949	1.16
$Q_r(\tilde{c}_1)$		1.10	1.03
$P_r(\tilde{d}_1)$			1.00
$Q_r(\tilde{d}_1)$			19.5
$P_r(\tilde{c}_2)$			.997
$Q_r(\tilde{c}_2)$			1.16

TABLE 5. The l^2 bounds for the different edge operators $\mathbf{E}_{H_L^{coif},\cdot}$ using Coiflet filters.

Table 4 and Table 5 clearly show the blowup of some of the edge-coefficients that results from using edge matrices. However, for the Daubechies filters the blowup effect almost exclusively affects the left-handside, while for the Coiflet filters the blowup effect does not seem to favour any side, and is on the whole more moderate. Anyway, this blowup will affect the numerical stability of the filtering-reconstruction procedure.

CHAPTER 5

Classification of Radar Signals Using Local Feature Extraction in the Space-Frequency Plane

1. Formulation of the problem

We consider the problem of separating two different distributions (classes) of electromagnetic radar signal sources from one another by doing a space-frequency analysis on the signals. Each distribution $D_n(\alpha)$ of signal sources will consist of a number of n coherent signal transmitters distributed randomly over n regularly spaced plane domains $\{\Omega_{n,i}(\alpha)\}_{i=1}^{n}$ with one and only one transmitter in each domain $\Omega(n,i)$. The domains may overlap.

$$\Omega_{n,i}(\alpha) = \{(r\cos\theta, r\sin\theta) : 1 \le r \le 10,\ 2\pi\frac{i}{n} \le \theta \le 2\pi\frac{i}{n} + \alpha\}.$$

A picture of $D_3(\cdot)$ and $D_4(\cdot)$ is shown in Figure 1, where the shaded areas are the $\{\Omega_{3,i}\}_{i=1}^{3}$ and the $\{\Omega_{4,i}\}_{i=1}^{4}$.

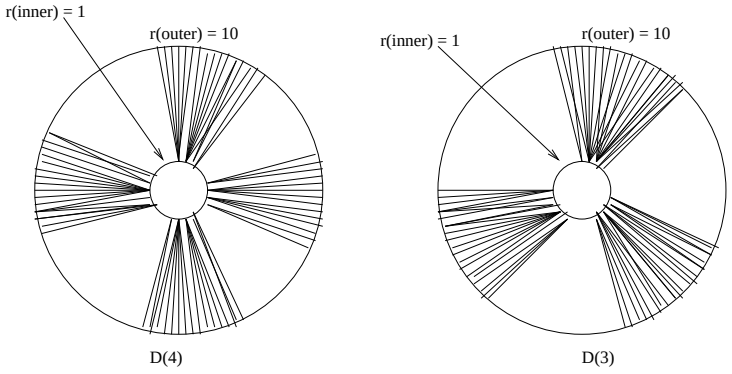

FIGURE 1. The domains of $D_4(\cdot)$ and $D_3(\cdot)$.

The analytical forms of the signals are solutions of the wave equation in three dimensions:

$$\frac{\partial^2\psi(\mathbf{x},t)}{\partial^2 t} - c^2\nabla^2\psi(\mathbf{x},t) = 0, \tag{5.1}$$

where $\nabla^2 = \frac{\partial^2}{\partial^2 x_1} + \frac{\partial^2}{\partial^2 x_2} + \frac{\partial^2}{\partial^2 x_3}$.

Making the ansatz $\psi(\mathbf{x},t) = \phi(\mathbf{x})e^{-i\omega t}$, we reduce the problem to the Helmholtz equation

$$\nabla^2\phi(\mathbf{x}) + k^2\phi(\mathbf{x}) = 0,\ k = \frac{\omega}{c}. \tag{5.2}$$

The fundamental solution of this equation may be expressed in polar coordinates as

$$\phi(r,\theta) = \phi(r) = A\frac{e^{ikr}}{r},\ A \text{ is a constant.}$$

Writing $\{\mathbf{p}_i\}_{i=1}^n$ for the locations of the signal sources in a signal source distribution $D_n(\cdot)$, we get the following analytical expression for the signal $s_n(\mathbf{x},t)$ corresponding to this distribution:

$$s_n(\mathbf{x},t) = e^{-ikt}\sum_{i=1}^{n}\frac{A_n(i)}{|\mathbf{x}-\mathbf{p}_i|}e^{ik|\mathbf{x}-\mathbf{p}_i|}. \tag{5.3}$$

We write

$$\mathbf{x} = (R\cos\theta, R\sin\theta),\ \mathbf{p}_i = (r_i\cos\theta_i, r_i\sin\theta_i).$$

Now, we assume R is large compared to the r_i's and make an expansion:

$$\begin{aligned}
|\mathbf{x}-\mathbf{p}_i| &= R\left[1 + \frac{r_i^2 - 2Rr_i(\cos\theta\cos\theta_i + \sin\theta\sin\theta_i)}{R^2}\right]^{1/2} \\
&\approx R + \frac{r_i^2}{2R} - r_i(\cos\theta\cos\theta_i + \sin\theta\sin\theta_i) \\
&= R + \frac{r_i^2}{2R} - r_i\cos(\theta-\theta_i),
\end{aligned}$$

then we can write

$$\begin{aligned}
s_n(R,\theta,t) &\approx e^{-ikt}\sum_{i=1}^{n}A_n(i)\frac{e^{ik(R+\frac{r_i^2}{2R}-r_i\cos(\theta-\theta_i))}}{R+\frac{r_i^2}{2R}-r_i\cos(\theta-\theta_i)} \\
&\approx \frac{e^{-ik(t-R)}}{R}\sum_{i=1}^{n}A_n(i)e^{ik(\frac{r_i^2}{2R}-r_i\cos(\theta-\theta_i))},
\end{aligned} \tag{5.4}$$

which is highly accurate if R is large enough.

For fixed $t = t'$, a plot of the magnitude of this function will be a smooth closed surface in $\mathbf{R}^3$ with a more or less spherical shape. This surface is called the wavefront at time t', and may be approximated locally "quite accurately" by a plane as illustrated in Figure 2, if the wavenumber k is not "very large".

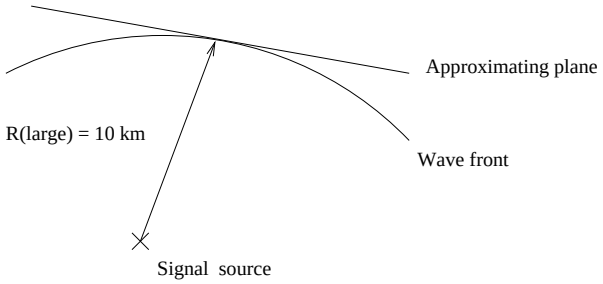

FIGURE 2. Local approximation of wavefront by a plane.

We now formulate our problem precisely: Is it possible, within some reasonable degree of accuracy, to separate $D_n(\alpha)$ from $D_m(\alpha)$, $n \neq m$, by simply doing constant phase measurements of the signals s_n, s_m in some small space interval far away from the signal source?

We will use the concepts and algorithms introduced in chapter 1 to give an answer to this question.

2. Experimental results

The results given in this section are very "experimental" in nature and were generated by implementing the Local Discriminant Basis Selection Algorithm described in Chapter 1.

By plotting the two most discriminating coordinates in the LDB for a collection of signals from two different distributions, we observe that one class is more or less centered around the origin in a relatively small area whereas the other class is spread over a much larger area around the origin. This phenomenon is to some degree documented in the scatter plots shown in appendix B. This indicates that it should be possible to achieve a reasonable degree of discrimination between the two classes by using suitable "hypersphere" surfaces in constructing the classifier. Define the set $B_{n,r}$ by

$$B_{n,r} = \{\mathbf{x} : x_1^2 + x_2^2 + \cdots + x_n^2 \leq r\}.$$

Let $\widehat{\mathbf{s}} \in \mathbf{R}^m$ be the point defined by the m most discriminating coordinates of some test signal $\mathbf{s} \in \mathbf{R}^n$. Given two distributions $D_x(\alpha) \neq D_y(\alpha)$ and m and r, we define the classifier $c_{m,r}$ by

$$c_{m,r}(\widehat{\mathbf{s}}) \in \begin{cases} D_x(\alpha) \text{ if } \widehat{\mathbf{s}} \in B_{m,r}, \\ D_y(\alpha) \text{ otherwise.} \end{cases}$$

We will concentrate on the problem of separating D_n from D_m when $|m-n| = 1$, since the results obtained in this case should get even better when $|m-n| > 1$. We compute the misclassification rate

on both the training dataset and the test dataset for each of four different real orthonormal bases: The "standard euclidean basis" (STB), a real Fourier-basis of type "discrete cosine basis" (DCB), the "best local cosine packet basis" (BLCPB) and the "best coiflet packet basis" (BCPB) using the coiflet of length 18. The discrete cosine basis will be of type IV, that is cosines evaluated at half integers in both time and frequency.

We note that the time-frequency localization properties of these four bases are very different: The elements of the standard euclidean basis are perfectly localized in time but possess no localization in frequency, the elements of the discrete cosine basis are perfectly localized in frequency and possess no localization in time, while the elements of BCPB and BLCPB possess localization in both time and frequency.

We use a training dataset of 250 signals from each class generated by randomly choosing points $\mathbf{p}_i \in \Omega_{n,i}(\alpha)$ $1 \leq n \leq 250$ in (5.3), and a test dataset of 2500 randomly generated signals of the same type from each class. We set the wavenumber k to 100 and the distance of observation R to 10^4. To not make the problem of extracting relevant features too easy for the LDB-algorithm, we equalize the maximum amplitude of the resultant signals s_n and s_{n+1} by setting $A_n(i) = \frac{1.0}{n}$. The real parts of the signals s_n, s_{n+1} were sampled $2^{11} = 2048$ times with period $\frac{2\pi}{16 \cdot k} \approx 4 \cdot 10^{-3}$ units, yielding a sampling interval at the point of observation of length $\frac{2\pi}{16 \cdot k} \cdot 2^{11} \approx 8.04$ units containing about $2^7 = 128$ oscillations of the signals s_n, s_{n+1}. The expansion depth in the packet bases is fixed to 8. The performance of the classifier $c_{m,r}$ is maximized subject to the conditions: $2 \leq m \leq 10$, $0.0 < r \leq 5.0$, where r is an integer multiplum of 2^{-7}.

Given D_n, D_{n+1}, we will study the performance of the LDB-algorithm and our particular classifier $c_{m,r}$ for the cases $\alpha = 30°$, $\alpha = 45°$, $\alpha = 60°$.

2.1. The case $D_2(\cdot), D_3(\cdot)$. The numbers given in boldface behind the error rates are the best m in the upper righthand corner and the best r in the lower righthand corner. The same m and r are applied to the test dataset, of course. The best test results for each α are written in bold face.

2.2. The case $D_3(\cdot), D_4(\cdot)$. The numbers given in boldface behind the error rates are the best m in the upper righthand corner and the best r in the lower righthand corner. The same m and r are applied to the test dataset, of course. The best test results for each α are written in bold face.

Method	Error rate (%)					
$c_{m,r}$ on	Training dataset			Test dataset		
	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$
STB	$35.80^{\mathbf{9}}_{\mathbf{1.27}}$	$38.40^{\mathbf{10}}_{\mathbf{1.39}}$	$21.40^{\mathbf{10}}_{\mathbf{1.43}}$	32.78	33.60	29.80
DCB	$37.60^{\mathbf{5}}_{\mathbf{1.07}}$	$39.60^{\mathbf{3}}_{\mathbf{0.77}}$	$48.60^{\mathbf{4}}_{\mathbf{0.28}}$	44.52	45.46	49.68
BCPB	$15.00^{\mathbf{10}}_{\mathbf{2.55}}$	$17.80^{\mathbf{8}}_{\mathbf{2.12}}$	$20.20^{\mathbf{10}}_{\mathbf{2.31}}$	16.62	22.84	25.20
BLCPB	$13.40^{\mathbf{10}}_{\mathbf{2.67}}$	$17.60^{\mathbf{10}}_{\mathbf{2.09}}$	$18.80^{\mathbf{10}}_{\mathbf{1.98}}$	**15.94**	**17.78**	**19.80**

TABLE 1. Numerical results from testing the performance of $c_{m,r}$ on $D_2(\cdot)$, $D_3(\cdot)$.

Method	Error rate (%)					
$c_{m,r}$ on	Training dataset			Test dataset		
	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$
STB	$31.20^{\mathbf{7}}_{\mathbf{0.89}}$	$31.40^{\mathbf{9}}_{\mathbf{1.16}}$	$40.00^{\mathbf{9}}_{\mathbf{1.09}}$	41.18	40.68	39.94
DCB	$38.40^{\mathbf{3}}_{\mathbf{0.92}}$	$45.00^{\mathbf{2}}_{\mathbf{0.95}}$	$40.40^{\mathbf{10}}_{\mathbf{2.31}}$	38.14	49.14	41.68
BCPB	$7.60^{\mathbf{7}}_{\mathbf{0.42}}$	$26.20^{\mathbf{2}}_{\mathbf{0.14}}$	$21.60^{\mathbf{7}}_{\mathbf{0.19}}$	**7.72**	**24.94**	**21.92**
BLCPB	$6.80^{\mathbf{8}}_{\mathbf{0.78}}$	$23.80^{\mathbf{6}}_{\mathbf{0.48}}$	$30.80^{\mathbf{9}}_{\mathbf{1.12}}$	10.56	25.20	31.74

TABLE 2. Numerical results from testing the performance of $c_{m,r}$ on $D_3(\cdot)$, $D_4(\cdot)$.

2.3. The case $D_4(\cdot), D_5(\cdot)$. The numbers given in boldface behind the error rates are the best m in the upper righthand corner and the best r in the lower righthand corner. The same m and r are applied to the test dataset, of course. The best test results for each α are written in boldface.

Method	Error rate (%)					
$c_{m,r}$ on	Training dataset			Test dataset		
	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$
STB	$34.40^{\mathbf{4}}_{\mathbf{0.63}}$	$35.80^{\mathbf{6}}_{\mathbf{0.76}}$	$43.00^{\mathbf{6}}_{\mathbf{0.63}}$	44.78	44.86	44.54
DCB	$35.80^{\mathbf{2}}_{\mathbf{0.61}}$	$42.00^{\mathbf{2}}_{\mathbf{1.01}}$	$41.20^{\mathbf{9}}_{\mathbf{1.48}}$	42.32	43.08	45.10
BCPB	$21.60^{\mathbf{10}}_{\mathbf{1.62}}$	$22.80^{\mathbf{2}}_{\mathbf{0.10}}$	$40.80^{\mathbf{2}}_{\mathbf{0.66}}$	23.62	25.32	43.96
BLCPB	$15.60^{\mathbf{10}}_{\mathbf{1.42}}$	$15.60^{\mathbf{5}}_{\mathbf{0.32}}$	$39.60^{\mathbf{4}}_{\mathbf{0.73}}$	**16.74**	**18.12**	**43.86**

TABLE 3. Numerical results from testing the performance of $c_{m,r}$ on $D_4(\cdot)$, $D_5(\cdot)$.

2.4. The case $D_5(\cdot), D_6(\cdot)$. The numbers given in boldface behind the error rates are the best m in the upper righthand corner and the best r in the lower righthand corner. The same m and r are applied to the test dataset, of course. The best test results for each α are written in boldface.

Method	Error rate (%)					
$c_{m,r}$ on	Training dataset			Test dataset		
	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$
STB	$32.60^{\mathbf{7}}_{\mathbf{0.65}}$	$43.00^{\mathbf{3}}_{\mathbf{0.58}}$	$32.20^{\mathbf{9}}_{\mathbf{0.82}}$	46.32	46.06	44.50
DCB	$41.20^{\mathbf{8}}_{\mathbf{1.30}}$	$35.80^{\mathbf{10}}_{\mathbf{1.74}}$	$38.40^{\mathbf{9}}_{\mathbf{1.69}}$	44.74	43.50	**44.22**
BCPB	$29.60^{\mathbf{4}}_{\mathbf{0.45}}$	$31.80^{\mathbf{5}}_{\mathbf{0.61}}$	$41.60^{\mathbf{10}}_{\mathbf{0.10}}$	35.46	33.80	49.18
BLCPB	$27.80^{\mathbf{6}}_{\mathbf{0.81}}$	$22.80^{\mathbf{5}}_{\mathbf{0.52}}$	$35.20^{\mathbf{10}}_{\mathbf{0.47}}$	**32.48**	**28.62**	46.72

TABLE 4. Numerical results from testing the performance of $c_{m,r}$ on $D_5(\cdot)$, $D_6(\cdot)$.

2.5. The case $D_{10}(\cdot), D_{20}(\cdot)$. The numbers given in boldface behind the error rates are the best m in the upper righthand corner and the best r in the lower righthand corner. The same m and r are applied to the test dataset, of course. The best test results for each α are written in boldface.

Method	Error rate (%)					
$c_{m,r}$ on	Training dataset			Test dataset		
	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$	$\alpha = 30°$	$\alpha = 45°$	$\alpha = 60°$
STB	$21.00^{\mathbf{8}}_{\mathbf{0.48}}$	$26.60^{\mathbf{10}}_{\mathbf{0.59}}$	$32.00^{\mathbf{10}}_{\mathbf{0.52}}$	26.62	29.22	29.08
DCB	$24.00^{\mathbf{10}}_{\mathbf{1.01}}$	$24.80^{\mathbf{10}}_{\mathbf{1.04}}$	$26.20^{\mathbf{9}}_{\mathbf{0.96}}$	25.90	25.16	**26.16**
BCPB	$20.00^{\mathbf{9}}_{\mathbf{0.67}}$	$28.40^{\mathbf{6}}_{\mathbf{0.56}}$	$39.60^{\mathbf{10}}_{\mathbf{0.71}}$	27.56	35.10	34.42
BLCPB	$22.80^{\mathbf{10}}_{\mathbf{1.06}}$	$17.80^{\mathbf{10}}_{\mathbf{1.08}}$	$20.20^{\mathbf{9}}_{\mathbf{0.97}}$	**24.50**	**24.86**	26.72

TABLE 5. Numerical results from testing the performance of $c_{m,r}$ on $D_{10}(\cdot)$, $D_{20}(\cdot)$.

3. Conclusion

Table 2.1 - Table 2.4 show that the classifier $c_{m,r}$ performs best on the coiflet packet and local cosine packet bases. Thus, we see that localization in *both* time/space and frequency is essential to provide the classifier with relevant features. We note that coiflets seem to be less "resistant" to overtraining than local cosines, they adapt too well to the training signals, that is.

APPENDIX A

$SO_2(\mathbf{R})$ elements for some orthogonal FIR filters.

Below we give the parameters $\alpha_k, 1 \leq k \leq L$, that determines the operators $W_2(\alpha_k)$ for 2 types of orthonormal filters of length L, Daubechies original shortest filters and Coiflet filters.

1. Daubechies shortest filters.

Daub 4

$$\alpha_1 = 0.5773502691$$
$$\alpha_2 = -0.2679491923$$

Daub 6

$$\alpha_1 = 0.4122865951$$
$$\alpha_2 = 1.831178514$$
$$\alpha_3 = -0.1058894200$$

Daub 8

$$\alpha_1 = 0.3222758836$$
$$\alpha_2 = 1.233150027$$
$$\alpha_3 = 3.856627874$$
$$\alpha_4 = -0.04600009616$$

Daub 10

$$\alpha_1 = 0.2651451339$$
$$\alpha_2 = 0.9398995872$$
$$\alpha_3 = 2.353886784$$
$$\alpha_4 = 7.508378888$$
$$\alpha_5 = -0.02083494630$$

Daub 12

$$\alpha_1 = 0.2255061720$$
$$\alpha_2 = 0.7643296306$$
$$\alpha_3 = 1.696013010$$
$$\alpha_4 = 4.114979257$$
$$\alpha_5 = 14.28573961$$
$$\alpha_6 = -0.009658362993$$

Daub 14

$$\alpha_1 = 0.1963287126$$
$$\alpha_2 = 0.6466065217$$
$$\alpha_3 = 1.333037518$$
$$\alpha_4 = 2.764759661$$
$$\alpha_5 = 7.035232916$$
$$\alpha_6 = 27.00281769$$
$$\alpha_7 = -0.004543409641$$

Daub 16

$$\alpha_1 = 0.1739238836$$
$$\alpha_2 = 0.5617332940$$
$$\alpha_3 = 1.103629937$$
$$\alpha_4 = 2.074598026$$
$$\alpha_5 = 4.380557848$$
$$\alpha_6 = 12.05139151$$
$$\alpha_7 = 49.52666172$$
$$\alpha_8 = -0.002443028170$$

Daub 18

$$\alpha_1 = 0.1561629731$$
$$\alpha_2 = 0.4973943657$$
$$\alpha_3 = 0.9452416623$$
$$\alpha_4 = 1.664172294$$
$$\alpha_5 = 3.114016860$$
$$\alpha_6 = 6.915226655$$
$$\alpha_7 = 20.60043019$$
$$\alpha_8 = 96.49772819$$
$$\alpha_9 = -0.001033336055$$

Daub 20

$$\alpha_1 = 0.1417287200$$
$$\alpha_2 = 0.4467987788$$
$$\alpha_3 = 0.8289658876$$
$$\alpha_4 = 1.394189716$$
$$\alpha_5 = 2.402966640$$
$$\alpha_6 = 4.635603726$$
$$\alpha_7 = 10.98508401$$
$$\alpha_8 = 35.63003753$$
$$\alpha_9 = 183.0054911$$
$$\alpha_{10} = -0.0004973444230$$

2. Coiflet filters.

Coif 6

$$\alpha_1 = -0.2152504427$$
$$\alpha_2 = 0.3779644639$$
$$\alpha_3 = -0.2152504427$$

Coif 12

$$\begin{aligned}
\alpha_1 &= -0.3952094767 \\
\alpha_2 &= -0.5625481503 \\
\alpha_3 &= 0.1165449040 \\
\alpha_4 &= 1.317233974 \\
\alpha_5 &= 6.198029576 \\
\alpha_6 &= -0.04396989341
\end{aligned}$$

Coif 18

$$\begin{aligned}
\alpha_1 &= -0.4874353702 \\
\alpha_2 &= -1.119071133 \\
\alpha_3 &= -0.2570708497 \\
\alpha_4 &= 0.1290348165 \\
\alpha_5 &= 0.4411074710 \\
\alpha_6 &= 2.215422179 \\
\alpha_7 &= 8.338120664 \\
\alpha_8 &= 15.03636438 \\
\alpha_9 &= -0.009120773147
\end{aligned}$$

Coif 24

$$\begin{aligned}
\alpha_1 &= -0.5476023581 \\
\alpha_2 &= -1.457533881 \\
\alpha_3 &= -0.7720754411 \\
\alpha_4 &= -0.1309276144 \\
\alpha_5 &= 0.1710353887 \\
\alpha_6 &= 0.2957793746 \\
\alpha_7 &= 0.8070747686 \\
\alpha_8 &= 3.126528296 \\
\alpha_9 &= 11.27596534 \\
\alpha_{10} &= 12.66598170 \\
\alpha_{11} &= 53.96686137 \\
\alpha_{12} &= -0.002000409650
\end{aligned}$$

Coif 30

$$\alpha_1 = -0.5914303923$$
$$\alpha_2 = -1.718001035$$
$$\alpha_3 = -1.195010469$$
$$\alpha_4 = -0.4056552189$$
$$\alpha_5 = -0.1316532923$$
$$\alpha_6 = 0.1205373016$$
$$\alpha_7 = 0.3671126852$$
$$\alpha_8 = 0.4678947012$$
$$\alpha_9 = 1.165968370$$
$$\alpha_{10} = 4.100416655$$
$$\alpha_{11} = 15.61099604$$
$$\alpha_{12} = 11.59905847$$
$$\alpha_{13} = 37.56973541$$
$$\alpha_{14} = 197.1316159$$
$$\alpha_{15} = -0.0004543371650$$

APPENDIX B

Edge matrices for orthonormal filters

The matrices $\mathbf{E}_{H_L,\cdot}$ are computed as follows: Each row in a $\mathbf{E}_{H_L,\cdot}$ corresponds to some $\tilde{c}$ or $\tilde{d}$ under the action of $\mathbf{E}_{H_L,\cdot}$ on $\mathbf{e}^j_\cdot$. The rows corresponding to $\tilde{c}$'s are uniquely determined by claiming polynomials up to degree $\frac{L}{2}-1$ mapping to polynomials of the same degree. The rows corresponding to $\tilde{d}$'s are not uniquely determined in this way, since claiming $\frac{L}{2}-1$ vanishing moments on each of the $\tilde{d}$'s results in the matrix $\mathbf{E}_{H_L,\cdot}$ becoming singular. This gives some "degrees of freedom" which were used to minimize the quantity $\|\mathbf{E}_{H_L,\cdot}\|_2 + \|\mathbf{E}^{-1}_{H_L,\cdot}\|_2$, that is the sum of the operator norms of the matrices, on some coarse grid.
Remarks:

- The edge-matrices $\mathbf{E}_{H_L,\cdot}$ operates on edge vectors $\mathbf{e}^j_\cdot$ resulting from non-normalized inner rotations.
- Entries less than 10^{-6} were set to zero.
- Superscripts, n, on the matrix entries denote multiplication by 10^{-n}.
- The matrix norm is the maximum column sum.

1. Edge matrices for H_6^d, H_6^{coif}

Applying $\mathbf{E}_{H_6^d,\cdot}$ or $\mathbf{E}_{H_6^{coif},\cdot}$ at edges leads to

- 1 vanishing moment on $\tilde{d}_0$.
- 1 vanishing moment on $\tilde{d}_{n/2-1}$.
- Polynomials up to and including degree 1 map to polynomials of the same degree.

Grid dimension = 1, Gridsize = .1

$$\mathbf{E}_{H_6^d,l} = \begin{pmatrix} 0 & 9.549704 \\ .7719849 & -1.35 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H_6^d,l} = \begin{pmatrix} .1831197 & 1.295362 \\ .1047153 & 0 \end{pmatrix}$$

$$\mathbf{E}_{H_6^d,r} = \begin{pmatrix} .7865626 & -.35 \\ 0 & 1.011213 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H_6^d,r} = \begin{pmatrix} 1.271355 & .4400402 \\ 0 & .9889118 \end{pmatrix}$$

$$\mathbf{E}_{H_6^{coif},l} = \begin{pmatrix} 0 & 4.861003 \\ .5221843 & -1.95 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H_6^{coif},l} = \begin{pmatrix} .7682189 & 1.915033 \\ .2057189 & 0 \end{pmatrix}$$

$$\mathbf{E}_{H^d_{coif},r} = \begin{pmatrix} .6742919 & -.35 \\ 0 & 1.046333 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H_6^{coif},r} = \begin{pmatrix} 1.483037 & .4960784 \\ 0 & .9557189 \end{pmatrix}$$

2. Edge matrices for H_8^d, H_8^{coif}

Applying $\mathbf{E}_{H_8^d,\cdot}$, $\mathbf{E}_{H_8^{coif},\cdot}$ leads to

- 1 vanishing moment on $\tilde{d}_0$.
- 2 vanishing moments on $\tilde{d}_1$ and $\tilde{d}_{n/2-1}$.
- Polynomials up to and including degree 2 map to polynomials of the same degree.

Remark: The coiflet of length 8 is the one we computed numerically in Table 3.
Left case: Grid dimension = 3, Gridsize = 1.
Right case: Grid dimension = 1, Gridsize = .1

$$\mathbf{E}_{H_8^d,l} = \begin{pmatrix} 5.702903 & -7.5 & -.5 \\ -.11143^3 & .1560114^3 & 21.78508 \\ 4.698626 & -7.876422 & 1.5 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H_8^d,l} = \begin{pmatrix} .8137842 & .7203228^1 & -.7748912 \\ .4854576 & .5171221^1 & -.5892172 \\ 0 & .4590297^1 & 0 \end{pmatrix}$$

$$\mathbf{E}_{H_8^d,r} = \begin{pmatrix} 0 & 4.124631 & -.1195278^1 \\ -1.186296 & 1.289667 & -.15 \\ 0 & 0 & 1.002116 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H_8^d,r} = \begin{pmatrix} .2635720 & -.8429597 & -.1230332 \\ .2424459 & 0 & .2891783^2 \\ 0 & 0 & .9978885 \end{pmatrix}$$

$$\mathbf{E}_{H_8^{coif},l} = \begin{pmatrix} -.1888304^1 & -5.5 & 10.5 \\ 0 & 0 & 2.666667 \\ -.8293807 & .4881421 & -1.5 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H_8^{coif},l} = \begin{pmatrix} -.1067955 & -.2563419 & -1.203287 \\ -.1814515 & .7167892 & .4131254^2 \\ 0 & .3750000 & 0 \end{pmatrix}$$

$$\mathbf{E}_{H_8^{coif},r} = \begin{pmatrix} 0 & -2.519102 & .7359046 \\ .7396251 & .8283321 & -.45 \\ 0 & 0 & 1.203777 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H_8^{coif},r} = \begin{pmatrix} .4445772 & 1.352037 & .2336397 \\ -.3969669 & 0 & .2426777 \\ 0 & 0 & .8307189 \end{pmatrix}$$

3. Edge matrices for H_{10}^d

Applying $\mathbf{E}_{H_8^d,\cdot}$ leads to

- 2 vanishing moments on $\tilde{d}_0$ and the $\tilde{d}_{n/2-1}$.
- 3 vanishing moments on $\tilde{d}_1$ and the $\tilde{d}_{n/2-2}$.
- Polynomials up to and including degree 3 map to polynomials of the same degree.

Left case : Grid dimension = 3, Gridsize = 1.
Right case : Grid dimension = 3, Gridsize = 1.

$$\mathbf{E}_{H^d_{10},l} = \begin{pmatrix} .1889731^1 & -.2458571^1 & -366.9168 & 306.9367 \\ -51.61952 & 50.72522 & .5 & 10.5 \\ .2652604^2 & -.3472944^2 & .1291323^2 & 48.01625 \\ -3.922730 & 5.645330 & -2.755913 & 1.5 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H^d_{10},l} = \begin{pmatrix} -.4205230^2 & -.6107834^1 & .2309330^1 & .5488057 \\ -.4252506^2 & -.4244118^1 & .1901781^1 & .5584793 \\ -.2725347^2 & 0 & .1742140^1 & 0 \\ 0 & 0 & .2082591^1 & .1007584^4 \end{pmatrix}$$

$$\mathbf{E}_{H^d_{10},r} = \begin{pmatrix} 10.45706 & -2.394158 & -5.5 & .5 \\ 0 & -.2181276 & 7.644881 & -.2776091^2 \\ 20.02188 & -35.08449 & 12.07371 & -.5 \\ 0 & .1417283^5 & 0 & 1.000434 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H^d_{10},r} = \begin{pmatrix} .1100016 & .9099426^1 & -.7506492^2 & -.5847607^1 \\ .6277529^1 & .9694300^1 & -.3278640^1 & -.4749111^1 \\ 0 & .1308065 & 0 & .3629740^3 \\ 0 & 0 & 0 & .9995661 \end{pmatrix}$$

4. Edge matrices for H^d_{12}, H^{coif}_{12}

Applying $\mathbf{E}_{H^d_{12},\cdot}$, $\mathbf{E}_{H^{coif}_{12},\cdot}$ leads to

- 2 vanishing moments on $\tilde{d}_0$.
- 3 vanishing moments on $\tilde{d}_1$ and $\tilde{d}_{n/2-1}$.
- 4 vanishing moments on $\tilde{d}_2$ and $\tilde{d}_{n/2-2}$.
- Polynomials up to and including degree 4 map to polynomials of the same degree.

Left case : Grid dimension = 6, Gridsize = 3.
Right case : Grid dimension = 3, Gridsize = 1.

$$\mathbf{E}_{H^d_{12},l} = \begin{pmatrix} -9.312776 & 16.23743 & -10 & -1 & 2 \\ -6.91477 & 5.863835 & -3.247505 & -1484.263 & 751.2569 \\ -47.71849 & 24.84963 & 8.319144 & -1 & 8 \\ -.8985454 & .7639290 & -.4245146 & .2903221 & 103.6496 \\ 11.10650 & -12.40799 & 9.908950 & -5.400903 & -1 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H^d_{12},l} = \begin{pmatrix} .1264939 & -.5806068^{3} & -.1332401^{1} & .4135265^{2} & .1388293 \\ .2045857 & -.8716865^{3} & .6821282^{2} & .3779404^{2} & .2006150 \\ .1144054 & -.8074226^{3} & .2347018^{1} & .3729639^{2} & .1965661 \\ -.2361415^{5} & -.6727232^{3} & .2410276^{5} & .4875633^{2} & -.1600175^{4} \\ .5729844^{5} & 0 & -.6966225^{4} & .9657503^{2} & .5300424^{3} \end{pmatrix}$$

$$\mathbf{E}_{H^d_{12},r} = \begin{pmatrix} .6614876^{4} & -.2352082^{3} & 62.56825 & -.2539657 & .2359844^{2} \\ -196.6045 & 298.3539 & -61.96023 & -10.5 & .5 \\ -.1009016^{5} & -.3181084^{4} & .3673955^{4} & 14.35707 & -.6760348^{3} \\ -139.7373 & 332.5374 & -210.6613 & 30.02657 & -.5 \\ -.3170654^{4} & .6220284^{4} & -.3292688^{4} & .4123728^{5} & 1.000093 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H^d_{12},r} = \begin{pmatrix} .2850594^{1} & -.1403871^{1} & -.3610553^{1} & .1259560^{1} & .1322425^{1} \\ .2210355^{1} & -.5899283^{2} & -.2128227^{1} & .8300054^{2} & .7032465^{2} \\ .1598260^{1} & 0 & .2826771^{3} & 0 & -.3750916^{4} \\ 0 & 0 & .6965203^{1} & 0 & .4709945^{4} \\ 0 & 0 & 0 & 0 & .9999068 \end{pmatrix}$$

$$\mathbf{E}_{H^{coif}_{12},l} = \begin{pmatrix} .3964447^{1} & -2.236082 & 2 & -1 & 2 \\ 6.924950 & 24.51832 & 22.46071 & -97.05257 & -16.09946 \\ .1434265 & -1.288120 & -.7670091 & 2 & 2 \\ 1.480799 & 5.242960 & 4.802957 & 10.23351 & 1.464548 \\ -.1388966 & -.4917635 & -.4504867 & -.9600 & 2 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H^{coif}_{12},l} = \begin{pmatrix} -.2325044 & .6560194^{1} & 2.279093 & .1082415^{1} & -1.526436 \\ -.1959253 & -.3493090^{3} & -.2975522 & .7647028^{1} & .4346684 \\ .2855577 & -.5141012^{2} & -.3778586 & .5405817^{1} & .1133166^{1} \\ 0 & -.6900798^{2} & 0 & .2532173^{1} & -.7409259^{1} \\ -.1446773^{5} & 0 & .6555962^{5} & .4388504^{1} & .4678566 \end{pmatrix}$$

$$\mathbf{E}_{H^{coif}_{12},r} = \begin{pmatrix} -6.463859 & 12.73118 & 6.045167 & -.1153122 & -.4242841^{2} \\ -17.25484 & -5.83848 & 24.97822 & -10.5 & .5 \\ 1.145926 & -2.257005 & 1.273759 & 6.245740 & -.2633039^{3} \\ -83.70978 & 164.8742 & -93.04824 & 9.198964 & -.5 \\ -.4281287 & .8432407 & -.4758912 & .4704796^{1} & .9993761 \end{pmatrix}$$

$$\mathbf{E}^{-1}_{H^{coif}_{12},r} = \begin{pmatrix} .8105815^{1} & -.4945805^{1} & -.6847096^{1} & -.9052881^{2} & .2054129^{1} \\ .8381187^{1} & -.2511078^{1} & -.3799309^{1} & -.1877050^{2} & .1196993^{1} \\ .7558504^{1} & 0 & .9794265^{2} & -.5689511^{2} & -.2523067^{2} \\ 0 & 0 & .1569448 & .2142764^{2} & .1113393^{2} \\ 0 & 0 & 0 & -.5104579^{2} & .9980704 \end{pmatrix}$$

APPENDIX C

Signal and Scatter Plots.

1. Plots for $D_2(45°)$, $D_3(45°)$

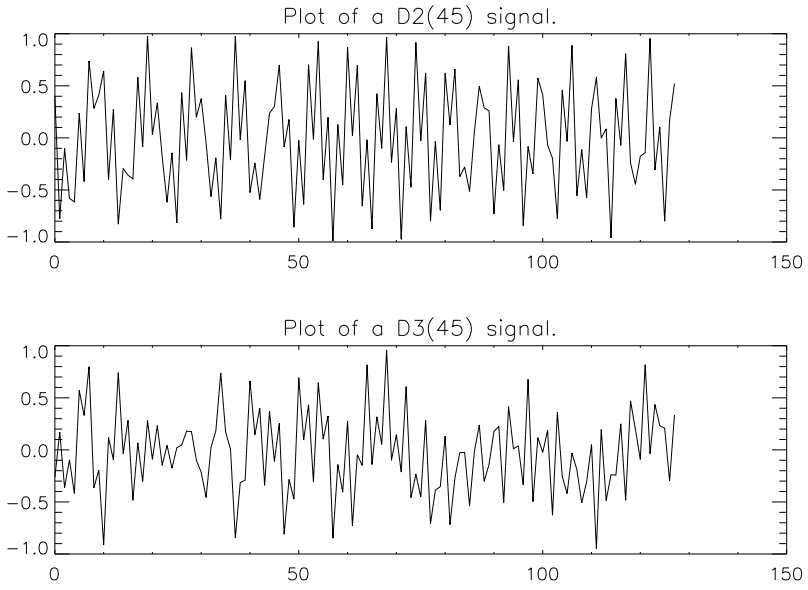

FIGURE 1. Plot of the first 128 samples of the signals s_2 and s_3.

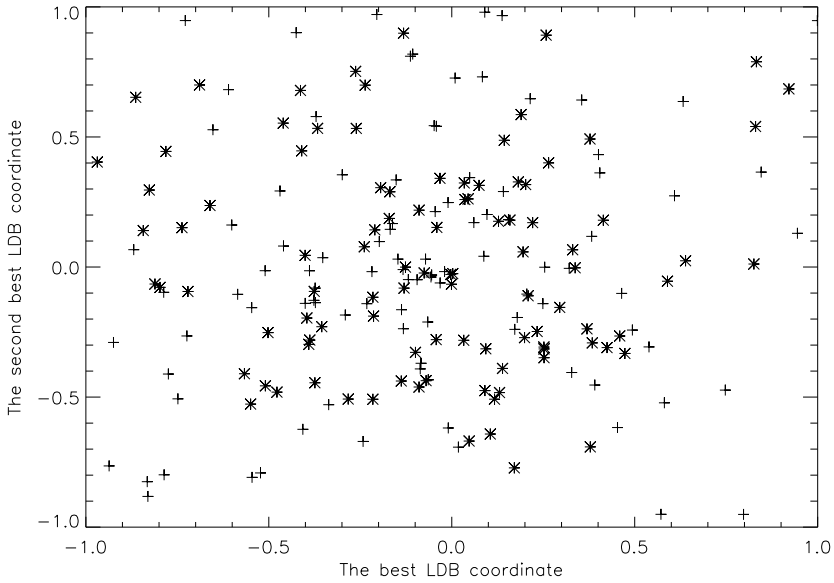

FIGURE 2. Scatter plot of the 2 most discriminating coordinates in the standard basis for 100 signals from each of the classes $D_2(45°)$ and $D_3(45°)$.

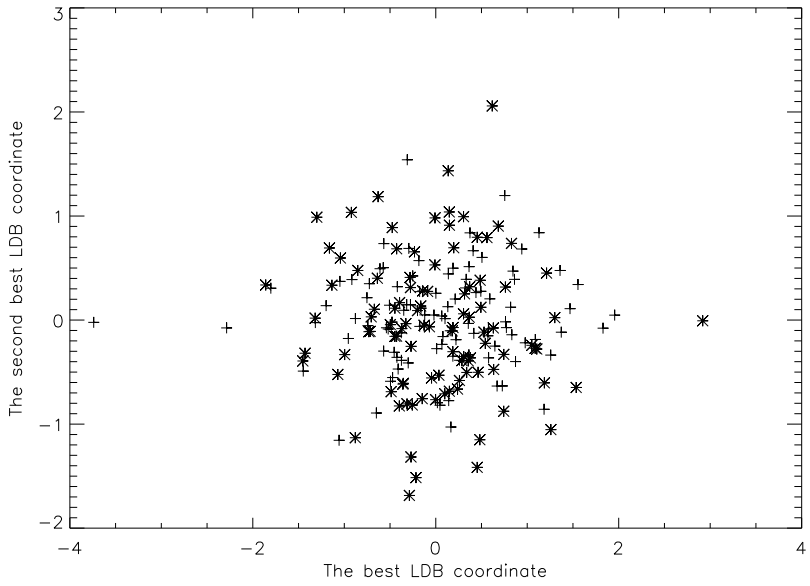

FIGURE 3. Scatter plot of the 2 most discriminating coordinates in the discrete cosine IV basis for 100 signals from each of the classes $D_2(45°)$ and $D_3(45°)$.

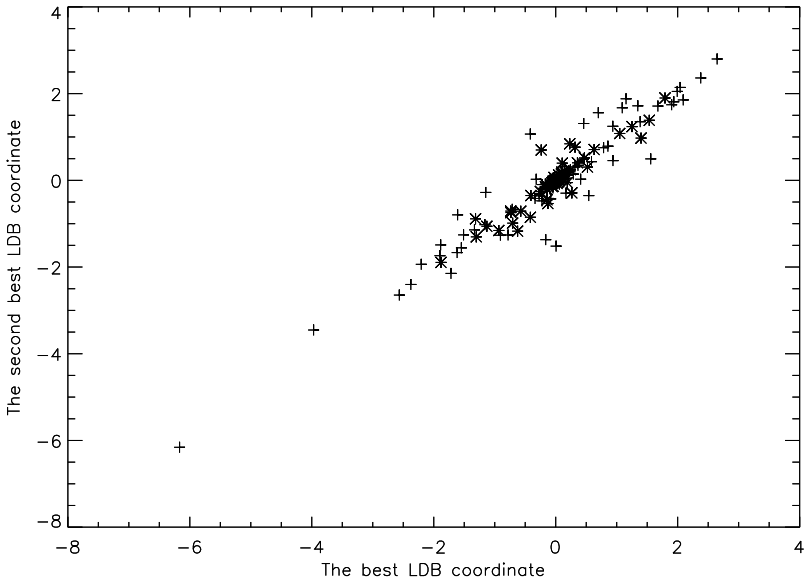

FIGURE 4. Scatter plot of the 2 most discriminating coordinates in the coiflet packet basis for 100 signals from each of the classes $D_2(45°)$ and $D_3(45°)$.

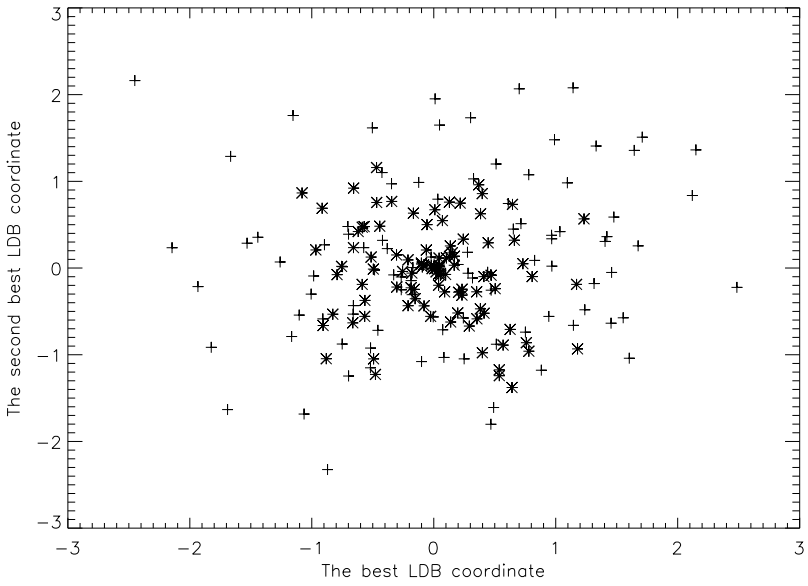

FIGURE 5. Scatter plot of the 2 most discriminating coordinates in the local cosine basis for 100 signals from each of the classes $D_2(45°)$ and $D_3(45°)$.

2. Plots for $D_3(45°)$, $D_4(45°)$

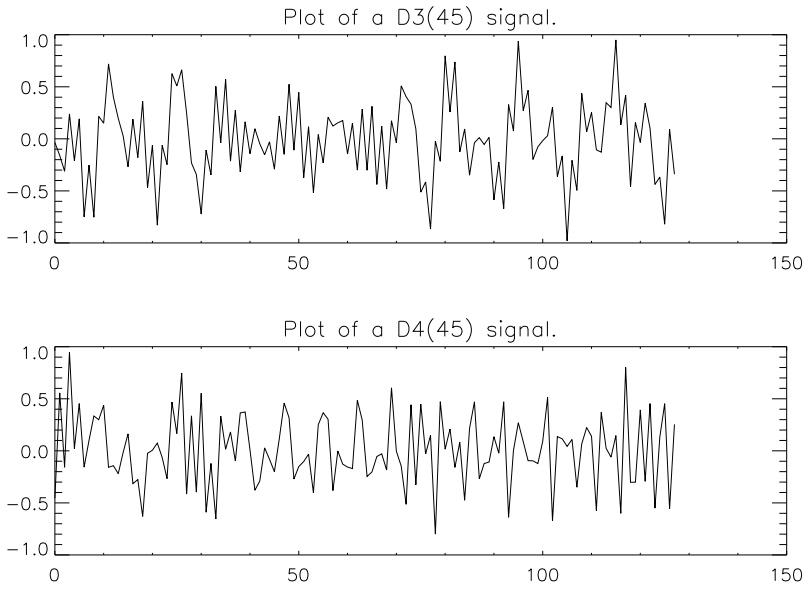

FIGURE 6. Plot of the first 128 samples of the signals s_3 and s_4.

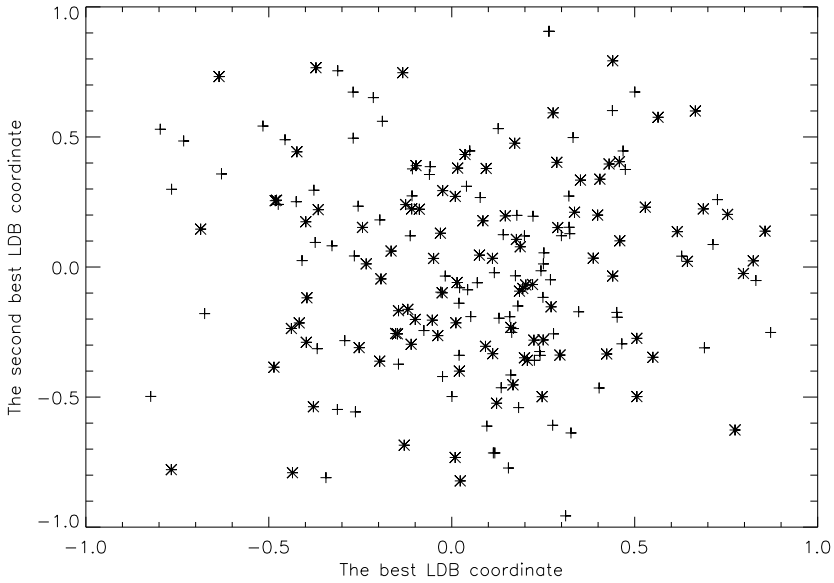

FIGURE 7. Scatter plot of the 2 most discriminating coordinates in the standard basis for 100 signals from each of the classes $D_3(45°)$ and $D_4(45°)$.

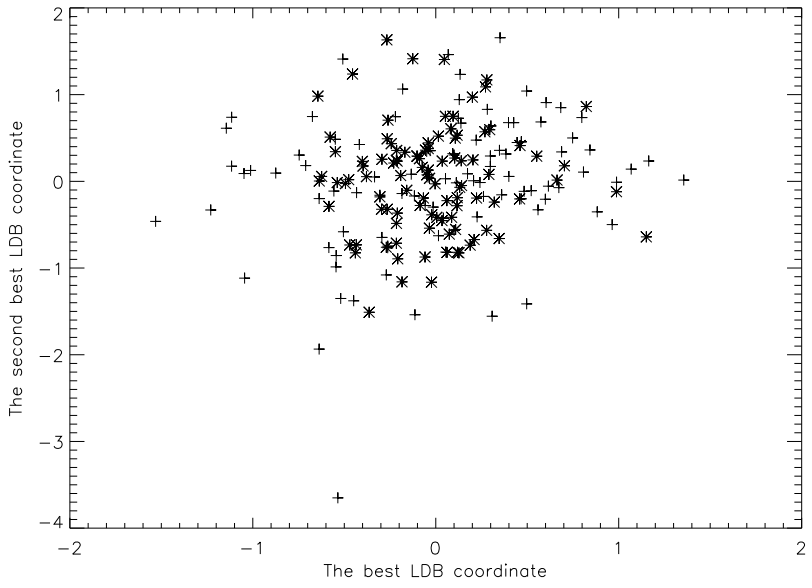

FIGURE 8. Scatter plot of the 2 most discriminating coordinates in the discrete cosine IV basis for 100 signals from each of the classes $D_3(45°)$ and $D_4(45°)$.

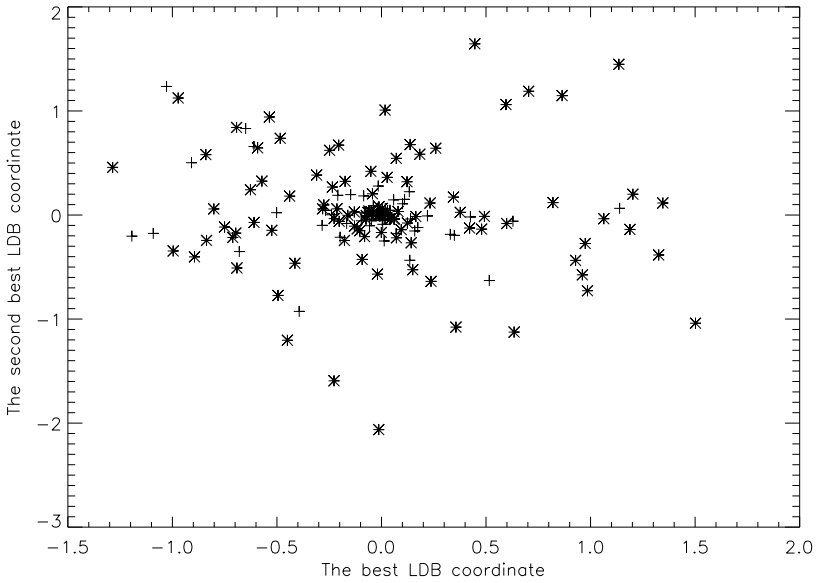

FIGURE 9. Scatter plot of the 2 most discriminating coordinates in the coiflet packet basis for 100 signals from each of the classes $D_3(45°)$ and $D_4(45°)$.

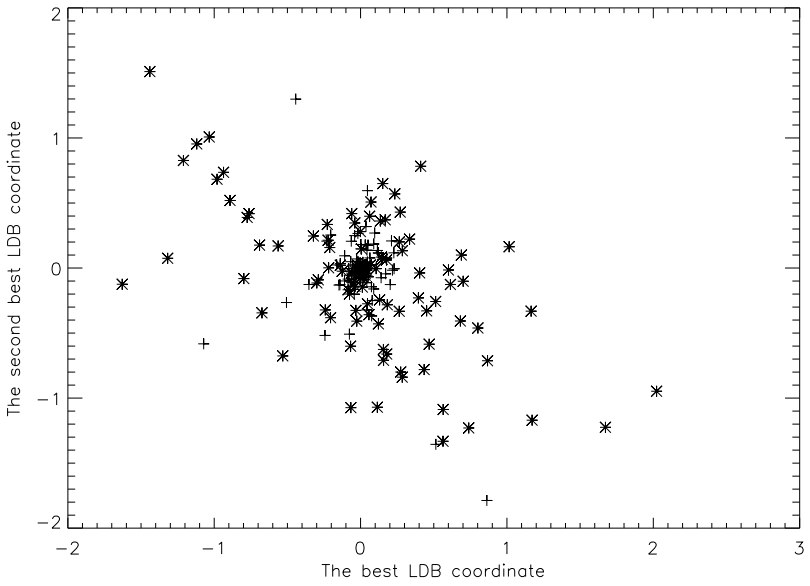

FIGURE 10. Scatter plot of the 2 most discriminating coordinates in the local cosine basis for 100 signals from each of the classes $D_3(45°)$ and $D_4(45°)$.

3. Plots for $D_4(45°)$, $D_5(45°)$

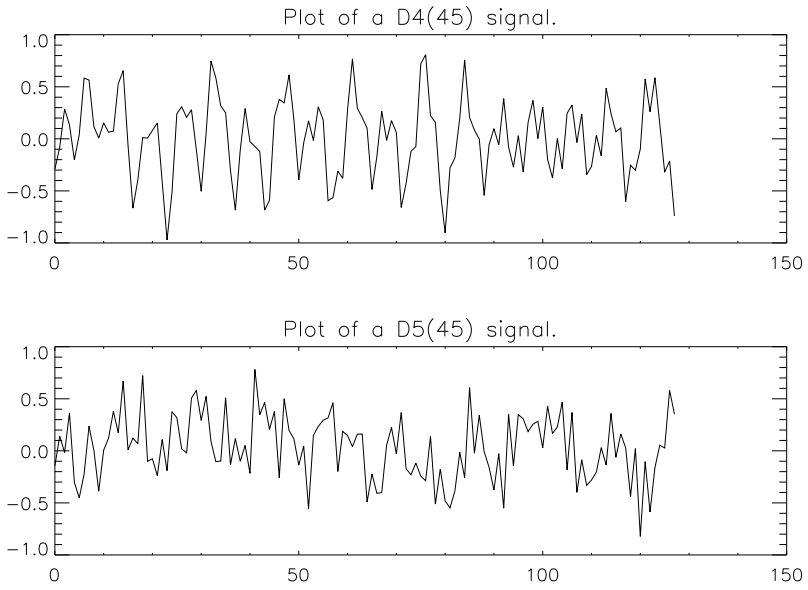

FIGURE 11. Plot of the first 128 samples of the signals s_4 and s_5.

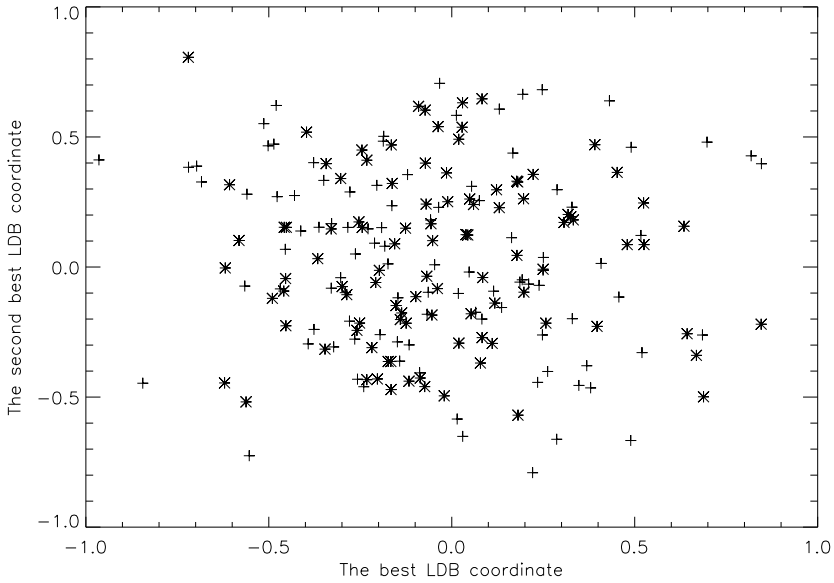

FIGURE 12. Scatter plot of the 2 most discriminating coordinates in the standard basis for 100 signals from each of the classes $D_4(45°)$ and $D_5(45°)$.

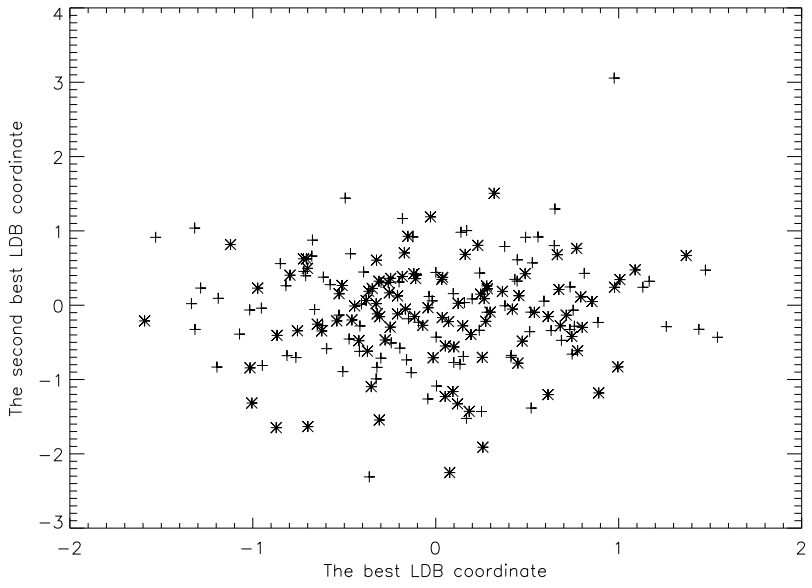

FIGURE 13. Scatter plot of the 2 most discriminating coordinates in the discrete cosine IV basis for 100 signals from each of the classes $D_4(45°)$ and $D_5(45°)$.

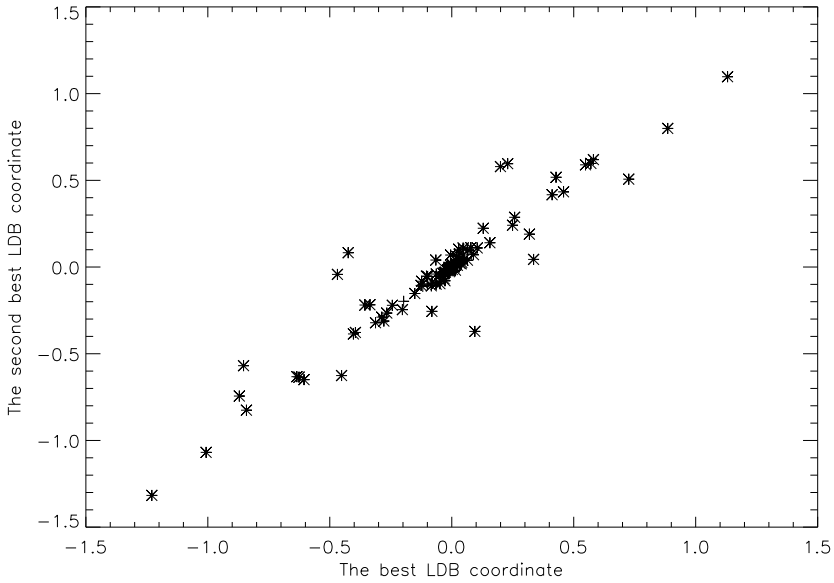

FIGURE 14. Scatter plot of the 2 most discriminating coordinates in the coiflet packet basis for 100 signals from each of the classes $D_4(45°)$ and $D_5(45°)$.

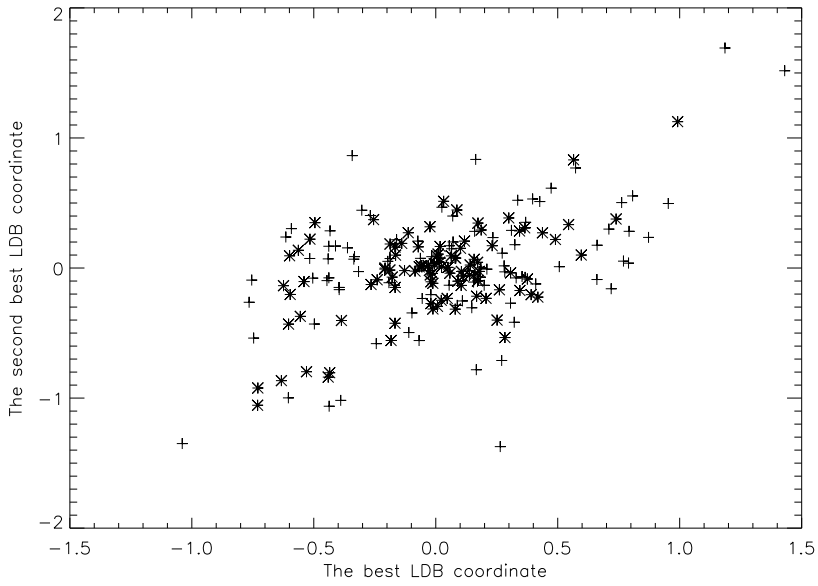

FIGURE 15. Scatter plot of the 2 most discriminating coordinates in the local cosine basis for 100 signals from each of the classes $D_4(45°)$ and $D_5(45°)$.

4. Plots for $D_5(45°)$, $D_6(45°)$

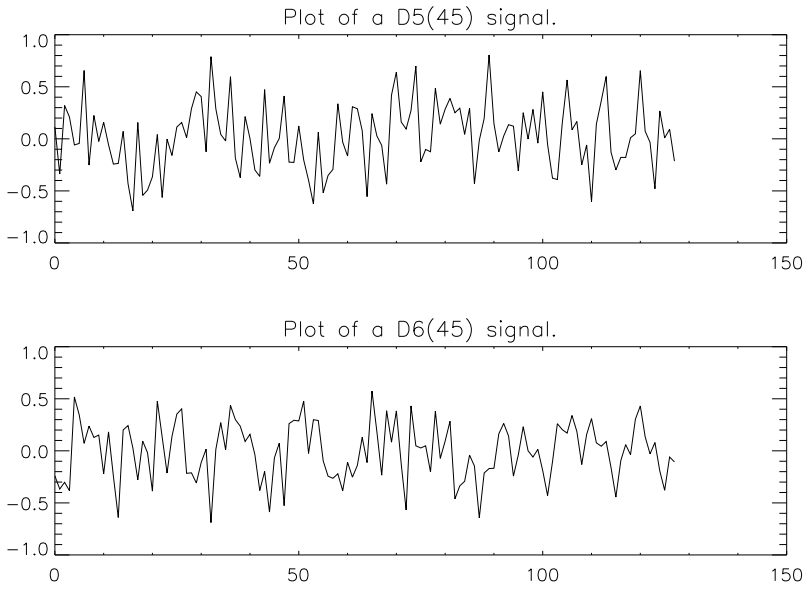

FIGURE 16. Plot of the first 128 samples of the signals s_5 and s_6.

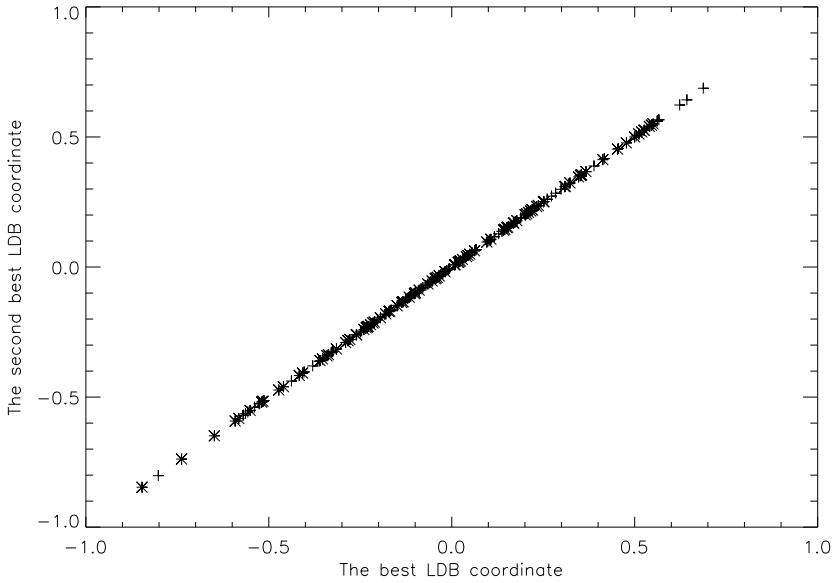

FIGURE 17. Scatter plot of the 2 most discriminating coordinates in the standard basis for 100 signals from each of the classes $D_5(45°)$ and $D_6(45°)$.

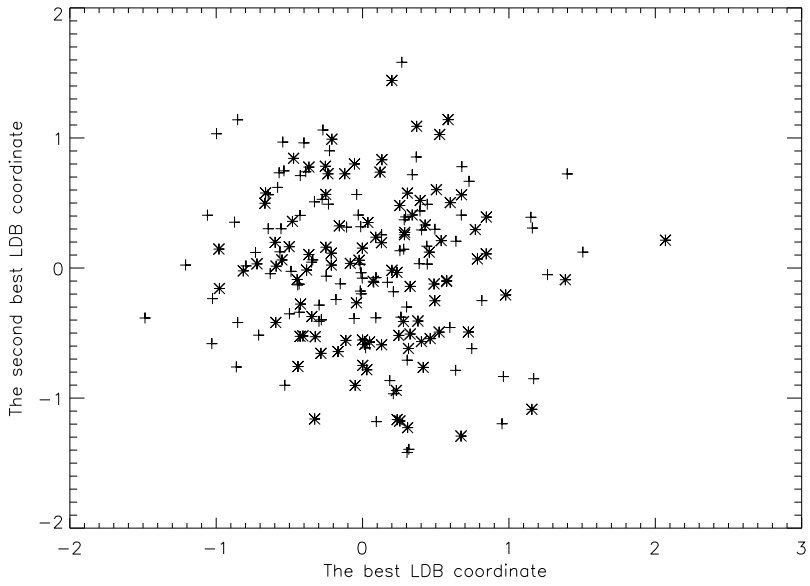

FIGURE 18. Scatter plot of the 2 most discriminating coordinates in the discrete cosine IV basis for 100 signals from each of the classes $D_5(45°)$ and $D_6(45°)$.

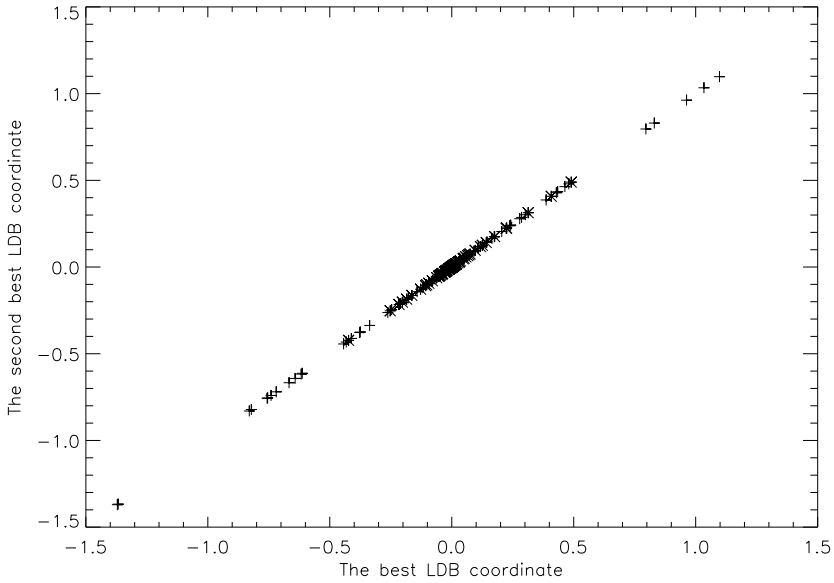

FIGURE 19. Scatter plot of the 2 most discriminating coordinates in the coiflet packet basis for 100 signals from each of the classes $D_5(45°)$ and $D_6(45°)$.

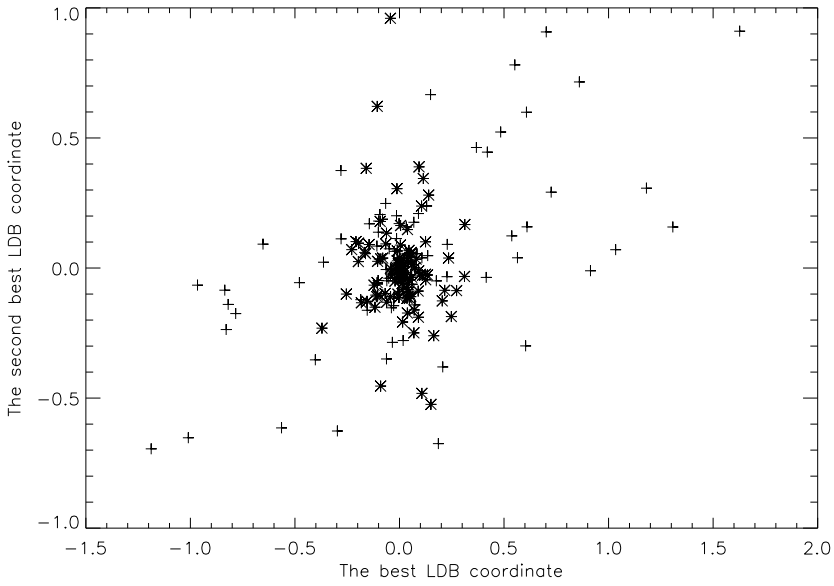

FIGURE 20. Scatter plot of the 2 most discriminating coordinates in the local cosine basis for 100 signals from each of the classes $D_5(45°)$ and $D_6(45°)$.

5. Plots for $D_{10}(45°)$, $D_{20}(45°)$

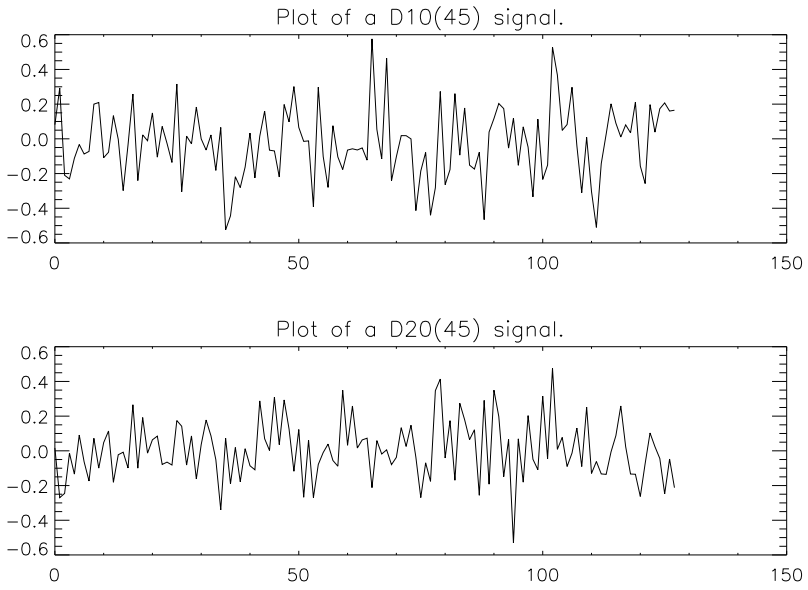

FIGURE 21. Plot of the first 128 samples of the signals s_{10} and s_{20}.

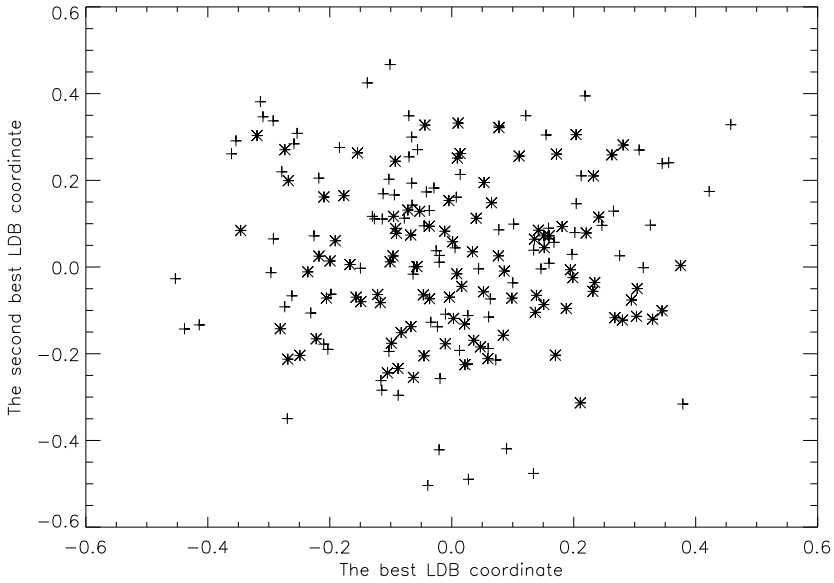

FIGURE 22. Scatter plot of the 2 most discriminating coordinates in the standard basis for 100 signals from each of the classes $D_{10}(45°)$ and $D_{20}(45°)$.

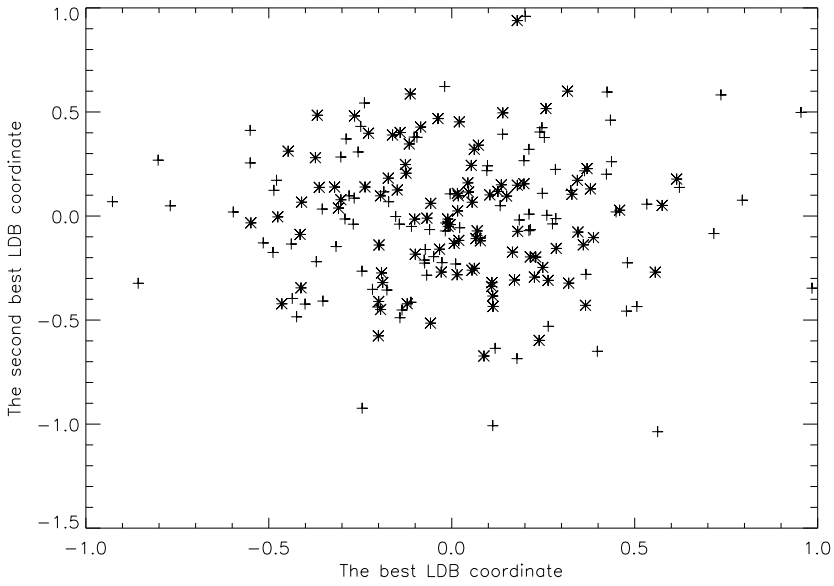

FIGURE 23. Scatter plot of the 2 most discriminating coordinates in the discrete cosine IV basis for 100 signals from each of the classes $D_{10}(45°)$ and $D_{20}(45°)$.

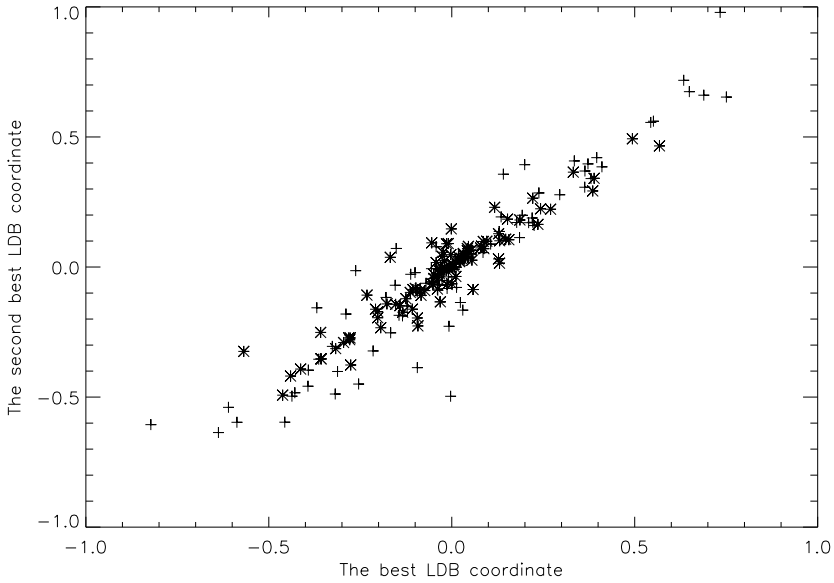

FIGURE 24. Scatter plot of the 2 most discriminating coordinates in the coiflet packet basis for 100 signals from each of the classes $D_{10}(45°)$ and $D_{20}(45°)$.

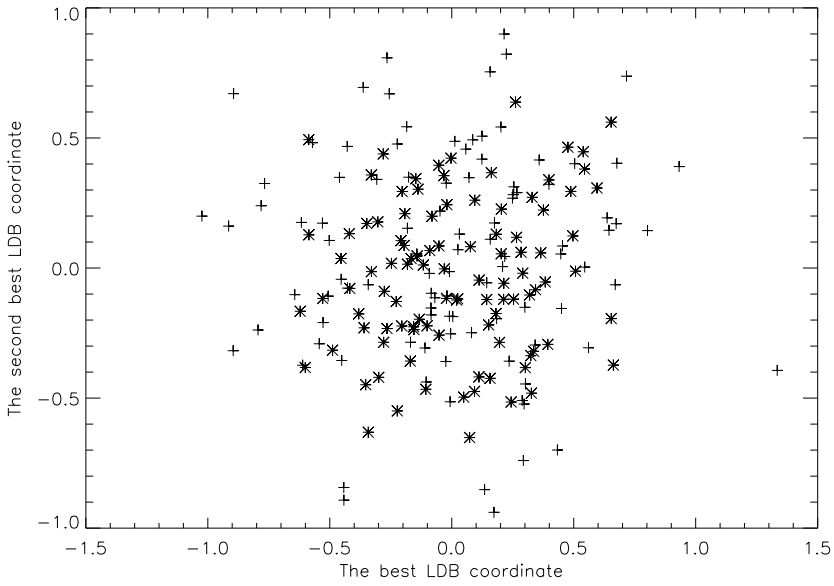

FIGURE 25. Scatter plot of the 2 most discriminating coordinates in the local cosine basis for 100 signals from each of the classes $D_{10}(45°)$ and $D_{20}(45°)$.

APPENDIX D

Computer Programs

All of the numerical work discussed in this paper was carried through in Ansi-C or Maple. In the three sections below we give the source code of all important transforms and algorithms used in this paper. The code in the first section was, except some slight modifications, copied from [**2**]. The code in the second and third section was generated by the author himself.

1. Borrowed Ansi C source code

```
/*********************************************************************/

/* Some useful functions: The discrete fourier, cosine/sine,
   local cosine/sine functions and their companions are copied
   (and somewhat adapted for our special use) from M.L.Wickerhauser:
   "Adapted Wavelet Analysis from Theory to Software". */

/*********************************************************************/

                                                                    10

/* typedef complex: Define a new data structure of type "complex"
   containing two members:

   RE = Floating point number describing the
   real part of a complex number.
   IM =  Floating point number describing the imaginary part
   of a complex number. */

typedef struct {
  double RE;                                                        20
  double IM;
} complex;


/* struct interval: Define a new data structure of type "interval",
   containing three members:

   ORIGIN = Floating point pointer to origin of an array of both
   positive and negative indexes.
   LEAST = Integer describing the least index of the array.         30
   FINAL = Integer describing the largest index of the array. */
```

```
struct interval {
  double *ORIGIN;
  int LEAST;
  int FINAL;
};

#define  CCMULRE(Z1,Z2)              (Z1.RE*Z2.RE-Z1.IM*Z2.IM)
#define  CCMULIM(Z1,Z2)              (Z1.RE*Z2.IM+Z1.IM*Z2.RE)
#define  CRRMULRE(Z,YRE,YIM)           (Z.RE*YRE-Z.IM*YIM)
#define  CRRMULIM(Z,YRE,YIM)           (Z.RE*YIM+Z.IM*YRE)
#define  max(X,Y)                   ((X > Y) ? X:Y)
#define  min(X,Y)                   ((X < Y) ? X:Y)
#define  rcf(T)                    rcfis(1,T)
#define  abtblock(N,L,B)             ((L)*(N)+(B)*((N)>>(L)))
#define  abtlength(N,L)              ((N)>>(L))

/* makeinterval: Allocate an interval data structure
   and assign its data array */

struct interval makeinterval(double *DATA, int LEAST, int FINAL)
{
  int LENGTH, K;

  struct interval SEG;

  LENGTH = 1+FINAL-LEAST;
  if (LENGTH > 0) {
    SEG.ORIGIN = (double *)calloc(LENGTH, sizeof(double));
    SEG.ORIGIN -= LEAST;
    if (DATA != NULL) {
      for (K = LEAST; K <= FINAL; K++) {
        SEG.ORIGIN[K] = DATA[K-LEAST];
      }
    }
  }
  SEG.LEAST = LEAST;
  SEG.FINAL = FINAL;
  return SEG;
}

/* freeinterval: Deallocate an interval data structure
   and its data array */

double *freeinterval(struct interval *SEG)
{
  if (SEG != NULL) {
    if ((SEG->ORIGIN) != NULL) {
      (SEG->ORIGIN) += SEG->LEAST;
```

```
      free(SEG->ORIGIN);
    }
    free(SEG);
  }
  return NULL;
}

/* br: Return the input integer bit-reversed */

int br(int N, int LOG2LEN)                                          90
{
  int U, J;

  U = N&1;
  for (J = 1; J < LOG2LEN; J++) {
    N >>= 1;
    U <<= 1;
    U += N&1;
  }
  return U;                                                         100
}

/* bitrevd: Permute to a disjoint array by bit-reversing
   the indices */

void bitrevd(complex *OUT, complex *IN, int Q)
{
  int M, N, U;

  M = 1<<Q;                                                         110
  for (N = 0; N < M; N++) {
    U = br(N, Q);
    OUT[U] = IN[N];
  }
}

/* bitrevi: Permute an array in place via index bit-reversal */

void bitrevi (complex *X, int Q)
{                                                                   120
  int M, N, U;

  complex TEMP;

  M = 1<<Q;
  for (N = 1; N < (M-1); N++) {
    U = br(N, Q);
    if (U > N) {
      TEMP = X[N];
```

```
      X[N] = X[U];
      X[U] = TEMP;
    }
  }
}

/* fftomega: Compute table of sines and cosines for DFT */

void fftomega(complex *W, int M)
{
  int K;

  double FACTOR;

  FACTOR = (double)(-PI/M);
  if (M < 0) {
    M = -M;
  }
  for (K = 0; K < M; K++) {
    W[K].RE = cos(K*FACTOR);
    W[K].IM = sin(K*FACTOR);
  }
}

/* fftproduct: Product of sparse matrices for DFT */

void fftproduct(complex *F, int Q, complex *W)
{
  int K, N, J, M, N1, B;

  complex TMP;

  N = 1<<Q;
  K = Q;
  while (K > 0) {
    K -= 1;
    N1 = N>>K;     /* Block size */
    M = N1/2;      /* Butterfly size */
    B = 0;
    while (B < N) {
      TMP.RE = F[B+M].RE;
      TMP.IM = F[B+M].IM;
      F[B+M].RE = F[B].RE - TMP.RE;
      F[B+M].IM = F[B].IM - TMP.IM;
      F[B].RE += TMP.RE;
      F[B].IM += TMP.IM;

      for (J = 1; J < M; J++) {
        TMP.RE = CCMULRE(F[B+M+J], W[J*(N/N1)]);
```

```
        TMP.IM = CCMULIM(F[B+M+J], W[J*(N/N1)]);
        F[B+M+J].RE = F[B+J].RE - TMP.RE;
        F[B+M+J].IM = F[B+J].IM - TMP.IM;
        F[B+J].RE += TMP.RE;
        F[B+J].IM += TMP.IM;
      }
      B += N1;
    }
  }
}

/* dct4omega: Compute table of sines and cosines for
   DCT-IV, DST-IV */

void dct4omega(double *COS, double *SIN, int M)
{
  int K;

  double FACTOR;

  FACTOR = PI/(2.0*M);
  for (K = 0; K < M; K++) {
    COS[K] = cos(K*FACTOR);
    SIN[K] = sin(K*FACTOR);
  }
}

/* dctnormal: Normalization for unitary L-point DCT
   and DST (T is log L) */

void dctnormal(double *Z, int L, int T)
{
  double NORM;

  int K;

  if ((T%2) != 0)
    NORM = 0.5/(1<<((T+1)/2)) ;
  else
    NORM = 0.5/sqrt(1<<(T+1));
  for (K = 0; K < L; K++) {
    Z[K] *= NORM;
  }
}

/* dct4: (in place) Unitary DCT-IV.
   OBS! Q is log of vectorlength */

void dct4(double *X, int Q)
```

```
{
  int N, K;
                                                                    230
  complex *F, *W, U, TMP;

  double *C, *S, *Y;

  N = 1<<Q;
  F = (complex *)calloc(2*N, sizeof (complex));
  W = (complex *)calloc(N, sizeof (complex));
  C = (double *)calloc(N, sizeof (double));
  S = (double *)calloc(N, sizeof (double));
  /* Y = (double *)calloc(N, sizeof (double)); */                   240

  dct4omega(C, S, N);
  F[0].RE = X[0];
  F[N].IM = X[N-1];

  for (K = 1; K < N; K++) {
    F[K].RE = X[K] * C[K];
    F[K].IM = -X[K] * S[K];
    F[2*N-K].RE = X[K-1]*C[K];
    F[2*N-K].IM = X[K-1]*S[K];                                      250
  }
  bitrevi(F, Q+1);
  fftomega(W, N);
  fftproduct(F, Q+1, W);
  U.RE = cos(-PI/(4.0*N));
  U.IM = sin(-PI/(4.0*N));
  TMP.RE = F[0].RE + CRRMULRE(F[2*N-1], C[1], S[1]);
  TMP.IM = F[0].IM + CRRMULIM(F[2*N-1], C[1], S[1]);
  X[0] = CCMULRE(TMP, U);
  TMP.RE = CRRMULRE(F[N-1], C[N-1], -S[N-1]) - F[N].IM;             260
  TMP.IM = CRRMULIM(F[N-1], C[N-1], -S[N-1]) + F[N].RE;
  X[N-1] = CCMULRE(TMP, U);

  for (K = 1; K < (N-1); K++) {
    TMP.RE = CRRMULRE(F[K], C[K], -S[K]) +
      CRRMULRE(F[2*N-K-1], C[K+1], S[K+1]);
    TMP.IM = CRRMULIM(F[K], C[K], -S[K]) +
      CRRMULIM(F[2*N-K-1], C[K+1], S[K+1]);
    X[K] = CCMULRE(TMP, U);
  }                                                                 270
  free(W);
  free(C);
  free(S);
  free(F);
  dctnormal(X, N, Q);
  /* return(Y); */
```

```
}

/* rcfis: Iterated sine rising cutoff function */

double rcfis(int N, double T)
{
  int I;

  if (T > -1.0) {
    if (T < 1.0) {
      for (I = 0; I < N; I++) {
        T = sin(0.5*PI*T);
      }
      T = sin(0.25*PI*(1.0 + T));
    }
    else
      T = 1.0;
  }
  else
    T = 0.0;
  return T;
}

/* rcfmidp: Rising cutoff function sampled between gridpoints */

void rcfmidp(struct interval R)
{
  int J;

  double X, DX;

  X = 0.5/(R.FINAL + 1.0);
  DX = 1.0/(R.FINAL + 1.0);
  for (J = 0; J <= R.FINAL; J++) {
    R.ORIGIN[J] = rcf(X);
    R.ORIGIN[-J-1] = rcf(-X);
    X += DX;
  }
}

/* fdcn: (midpoint) Fold disjoint cosine negative */

void fdcn(double *ONEG, int STEP, double *INEG,
          double *IPOS, int N, struct interval RISE)
{
  int K;

  for (K = -N; K <= (RISE.LEAST - 1); K++) {
```

```
    *(ONEG+K*STEP) = *(INEG+K);
  }
  for (K = RISE.LEAST; K <= -1; K++) {
    *(ONEG+K*STEP) = RISE.ORIGIN[-1-K]*(*(INEG+K)) -
      RISE.ORIGIN[K]*(*(IPOS-1-K));
  }
}


/* fdcp: (midpoint) Fold disjoint cosine positive */

void fdcp(double *OPOS, int STEP, double *INEG,
          double *IPOS, int N, struct interval RISE)
{
  int K;

  for (K = 0; K <= RISE.FINAL; K++) {
    *(OPOS+K*STEP) = RISE.ORIGIN[K]*(*(IPOS+K)) +
      RISE.ORIGIN[-1-K]*(*(INEG-1-K));
  }
  for (K = (RISE.FINAL + 1); K <= (N-1); K++) {
    *(OPOS+K*STEP) = *(IPOS+K);
  }
}

/* fdsn: (midpoint) Fold disjoint sine negative */

void fdsn(double *ONEG, int STEP, double *INEG,
          double *IPOS, int N, struct interval RISE)
{
  int K;

  for (K = -N; K <= (RISE.LEAST - 1); K++) {
    *(ONEG+K*STEP) = *(INEG+K);
  }
  for (K = RISE.LEAST; K <= -1; K++) {
    *(ONEG+K*STEP) = RISE.ORIGIN[-1-K]*(*(INEG+K)) +
      RISE.ORIGIN[K]*(*(IPOS-1-K));
  }
}

/* fdsp: (midpoint) Fold disjoint sine positive */

void fdsp(double *OPOS, int STEP, double *INEG,
          double *IPOS, int N, struct interval RISE)
{
  int K;

  for (K = 0; K <= RISE.FINAL; K++) {
```

```
    *(OPOS+K*STEP) = RISE.ORIGIN[K]*(*(IPOS+K)) -
      RISE.ORIGIN[-1-K]*(*(INEG-1-K));
  }
  for (K = (RISE.FINAL + 1); K <= (N-1); K++) {
    *(OPOS+K*STEP) = *(IPOS+K);
  }                                                          380
}

/* localcosine: (in place) (adapted dyadic) local cosine analysis
   (with) fixed folding applying a fixed cutoff function "RISE"
   sampled between gridpoints resulting in L+1 different
   representations of the signal.

   We have:

   N = The length of the signal to be analysed.              390

   L = The depth of expansion in the dictionary of bases.

   PARENT = Pointer to array of length N*(L+1) containing
   the signal to be analysed in its first N locations. */

void localcosine(double *PARENT, int N, int L)
{
  int NP, NC, LEVEL, PBLOCK, RADIUS;
                                                             400
  double *MIDP, *CHILD;

  struct interval RISE;

  RADIUS = (N>>(L+1))-1;
  if (RADIUS <= 0)
    printf("Negative folding radius in lcaff\n");

  RISE = makeinterval(NULL,-RADIUS, RADIUS-1);
  rcfmidp(RISE);                                             410
  NP = N;

  for (LEVEL = 0; LEVEL < L; LEVEL++) {
    NC = NP/2;

    for (PBLOCK = 1; PBLOCK <= (1<<LEVEL); PBLOCK++) {
      MIDP = PARENT + NC;
      CHILD = MIDP + N;
      fdcn(CHILD, 1, MIDP, MIDP, NC, RISE);
      fdcp(CHILD, 1, MIDP, MIDP, NC, RISE);                  420
      dct4(PARENT, intlog2(NP));
      PARENT += NP;
    }
```

```
    NP = NC;
  }
  for (PBLOCK = 1; PBLOCK <= (1<<L); PBLOCK++) {
    dct4(PARENT, intlog2(NP));
    PARENT += NP;
  }
  freeinterval(&RISE);                                          430

}
```

2. Author's Ansi C source code

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>

/* basisnode: Define a new data structure of type basisnode
   containing five members:

   LEVEL = The level coordinate of a subspace in the dictionary tree.
   BLOCK = The position of a subspace counted from the left.
   POS = The coordinate of a basisfunction in the subspace.
   COST = The cost/discrimination number.
   TAG = YES or NO. */

struct basisnode {
  int LEVEL;
  int BLOCK;
  int POS;
  double COST;
  int TAG;
};

#define  YES                          1
#define  NO                           0
#define  PI                           3.1415926535
#define  SQ2                          1.4142135623
#define  SQH                          0.7071067811
#define  MAXINT                         2147483647
#define  DIVISOR                      127773
#define  HIGH                         2836
#define  LOW                          16807
#define  SEEDFACTOR                       76928675
#define  CLASS_1                      10
#define  CLASS_2                      20
#define  K_1                          pow(10.0,2.0)
#define  K_2                          pow(10.0,2.0)
#define  A_1                          0.100
#define  A_2                          0.050
#define  RDIST                          10000
#define  expand                       localcosine
#define  s_measure(IN_1,IN_2,N)           lp_norm_p(IN_1,IN_2,N,2.0)
#define  makesignal(IN,N,K,A,M)           make_samples_of_cosine_signal(
IN,N,K,A,RDIST,16,M,1.0/8.0,PI/4)

/* Allocate a basisnode data structure an assign its content */
```

```
struct basisnode *makebasisnode(int LEVEL, int BLOCK,
                                int POS, double COST, int TAG)
                                                                    50
{
  struct basisnode *BASISNODE;

  BASISNODE = (struct basisnode *)malloc(sizeof(struct basisnode));

  BASISNODE->LEVEL = LEVEL;
  BASISNODE->BLOCK = BLOCK;
  BASISNODE->POS = POS;
  BASISNODE->COST = COST;
  BASISNODE->TAG = TAG;                                             60

  return(BASISNODE);
}

/* Deallocate an array of length N of pointers
   to basisnode data structures */

void freebasisnodes(struct basisnode **A, int N)
{
  int K;                                                            70

  for (K = 0; K < N; K++) {
    free(A[K]);
  }
  free(A);
}

/* ladder: (disjoint) filter a real array of length N into
   lowpass and highpass-coefficients by the "rotation" algorithm
   applying an orthonormal filter of length 2*L, periodizing        80
   at endpoints.

   We have:

   IN = Pointer to array of numbers to be filtered.

   OUT = Pointer to array that results from filtering IN.

   ALPHA = Pointer to array of "rotation" coefficients including
   a normalization factor.                                          90

   N = The length of IN = The length of OUT.

   L = Half the length of the filter. */

void ladder(double *OUT, double *IN,
```

```
            double *ALPHA, int N, int L)
{
  int LEVEL, TRANSLATION, K;
                                                                    100
  double *WORK;

  WORK = (double *)calloc(N, sizeof(double));
  for (K = 0; K < N; K++)
    WORK[K] = IN[K];

  for (LEVEL = 0; LEVEL < L; LEVEL++) {
    TRANSLATION = (LEVEL%2);

    for (K = 0; K < N/2; K++) {                                     110
      *(OUT+TRANSLATION + 2*K) =
        WORK[TRANSLATION + 2*K] - ALPHA[LEVEL]*
        WORK[(TRANSLATION + 1 + 2*K)%N];
      *(OUT+(TRANSLATION + 1 + 2*K)%N) =
        ALPHA[LEVEL]*WORK[TRANSLATION + 2*K] +
        WORK[(TRANSLATION + 1 + 2*K)%N];

      WORK[TRANSLATION + 2*K] =
        *(OUT+TRANSLATION + 2*K);
      WORK[(TRANSLATION + 1 + 2*K)%N] =                             120
        *(OUT+(TRANSLATION + 1 + 2*K)%N);
    }
  }
  for (K = 0; K < N/2; K++) {

    /* Store the lowpass and highpass coefficients in
       separate blocks. */

    if (L%2 == 0) {
      *(OUT+K) = WORK[2*K +1]*ALPHA[L];        /* c's */            130
      *(OUT+K+N/2) = WORK[2*K]*ALPHA[L];       /* d's */
    }

    else {
      *(OUT+K) = WORK[2*K]*ALPHA[L];           /* c's */
      *(OUT+K+N/2) = WORK[2*K+1]*ALPHA[L];     /* d's */
    }
  }
  free(WORK);
                                                                    140
}

/* waveletpacket: (in place) waveletpacket analysis resulting in L+1
   different representations of the signal.
```

```
  We have:

  N = The length of the signal to be analysed.

  L = The depth of expansion in the dictionary of bases.

  PARENT = Pointer to array of length N*(L+1) containing
  the sampled lowpass coefficients of the signal to be analysed
  in its first N locations.

  ALPHA = Pointer to array containing in the following order:

  1. Half the length of the orthonormal filter.
  2. The rotation coefficients corresponding to the filter.
  3. A scaling factor. */

void waveletpacket(double *PARENT, int N, int L)
{

  FILE *FILTER;

  int LEVEL, BLOCK, FILEN, NP, M, K;

  double *CHILD, *ALPHA;

  if(( FILTER = fopen("/users/stud/eirikf/programs/hovedfag/
  filters/laddercoif18","r"))
     == NULL) {

    printf("ERROR in opening FILTERfile\n");
    exit(1);
  }

  fscanf(FILTER, "%d", &FILEN);
  ALPHA = (double *)calloc(FILEN+1, sizeof(double));
  for (K = 0; K <= FILEN; K++) {
    fscanf(FILTER, "%lf", &ALPHA[K]);
  }
  fclose(FILTER);

  for (LEVEL = 0; LEVEL < L; LEVEL++) {
    NP = N>>LEVEL;

    for (BLOCK = 0; BLOCK < (1<<LEVEL); BLOCK++) {
      CHILD = PARENT+N;
      ladder(CHILD, PARENT, ALPHA, NP, FILEN);
      PARENT += NP;
    }
  }
```

```
  free(ALPHA);

}

/* my_qsort: sort v[left]...v[right] into decreasing order.
   (This function is copied from Kernighan/Ritchie:                    200
   "The Ansi C programming language" second edition p.120.) */

void my_qsort(void *v[], int left, int right,
          int (*comp)(void *, void *))
{
  int i, last;

  void my_swap(void *v[], int, int);

  if (left >= right)                                                  210
    return;

  my_swap(v, left, (left + right)/2);
  last = left;
  for (i = left+1; i <= right; i++) {
    if ((*comp)(v[i], v[left]) < 0)
      my_swap(v, ++last, i);
  }
  my_swap(v, left, last);
  my_qsort(v, left, last-1, comp);                                    220
  my_qsort(v, last+1, right, comp);
}

/* my_swap: Exchange pointers within an array */

void my_swap(void *v[], int i, int j)
{
  void *temp;

  temp = v[i];                                                        230
  v[i] = v[j];
  v[j] = temp;
}

/* comp_basisnode_by_cost: Compare the discrimination measures
   of two basisnode data structures */

int comp_basisnode_by_cost(struct basisnode *X,
                           struct basisnode *Y)
{                                                                     240
  if ((X->COST) > (Y->COST))
    return(-1);
  else if ((X->COST) < (Y->COST))
```

```
    return(1);
  else
    return(0);
}

/* my_rand: Generate a pseudo random floating point
   number between 0 and 1. */                                    250

double my_rand(void)
{
  int high, low, temp;

  static int SEED;

  if (SEED < 1 || SEED > MAXINT)
    SEED = ( (int) time(NULL)*SEEDFACTOR ) % MAXINT;
                                                                  260
  high = SEED/DIVISOR;
  low = SEED % DIVISOR;
  temp = LOW*low - HIGH*high;
  SEED = (temp > 0 ? temp : temp + MAXINT);
  return((double)SEED/MAXINT);
}

/* lp_norm_p: Compute the (l^P difference)^P of two arrays
   of length N */
                                                                  270
double lp_norm_p(double *IN_1, double *IN_2, int N, double P)
{
  double T, X;

  int K;

  T = 0;
  if (IN_2 == NULL) {
    for (K = 0; K < N; K++) {
      X = fabs(*(IN_1+K));                                        280
      if (X > 0)
        T +=exp(P*log(X));
    }
  }

  else {
    for (K = 0; K < N; K++) {
      X = fabs(*(IN_1+K)-(*(IN_2+K)));
      if ( X > 0)
        T += exp(P*log(X));                                       290
    }
  }
```

```
  return T;
}

/* modf_2PI: Compute remainder of input modulo 2*PI.
   (This is because of the bad accuracy of the
   standard math trigonometric functions for large aruments) */

double modf_2PI(double X)                                          300
{
  int K;

  double T, S;

  T = X;

  S = 2.0*PI;
  if (T > 0) {
    while (T > 0)                                                  310
      T -= S;

    T += S;
  }
  else {
    while (T < 0)
      T += S;
  }
  return(T);
                                                                   320
}

/* set_tag_to_zero: Set the tags of all children nodes of a
   basisnode data type, to NO. */

void set_tag_to_zero(struct basisnode **BASIS, int L,
                     int LEVEL, int BLOCK)
{
  int PARENT, LEFTCHILD, RIGHTCHILD;
                                                                   330
  PARENT = (1<<LEVEL) - 1 + BLOCK;
  LEFTCHILD = 2*PARENT + 1;
  RIGHTCHILD = LEFTCHILD + 1;
  BASIS[LEFTCHILD]->TAG = NO;
  BASIS[RIGHTCHILD]->TAG = NO;
  if (LEVEL < (L - 1)) {
    set_tag_to_zero(BASIS, L, LEVEL+1, 2*BLOCK);
    set_tag_to_zero(BASIS, L, LEVEL+1, 2*BLOCK + 1);
  }
                                                                   340
}
```

```
/* make_samples_of_cosines: Returns an array of samples of
   a specific (cosine)function.

   We have:

   SIGNAL = Array of length N*L

   N = The length of the sampling.                                  350

   K = The wavenumber of the transmitted wave(s).

   A = Amplitude (factor) of signal.

   R = The distance from origo to the interval arc of sampling.

   P = The number of samples pr. wavelength.

   M = The number of signal transmitters                            360

   T = Starting point of sampling.

   W = Width of sector containing the signal emitter
   as a fraction of 2.0PI. */

void make_samples_of_cosine_signal(double *SIGNAL, int N, double K,
                                   double A, double R, int P, int M,
                                   double W, double T)
{                                                                   370
  int I, J, L;

  double RADIUS, ANGLE, SAMPL, AMP, ARG, X, Y, *WORK;

  SAMPL = 2.0*PI/(K*P);

  WORK = (double *)calloc(N, sizeof(double));

  RADIUS = 0;
  L = 1;                                                            380

  for (J = 1; J <= M; J++) {
    while (RADIUS < 1.0)
      RADIUS = 10.0*my_rand();

    ANGLE = 2.0*PI*(L*1.0/M + my_rand()*W);
    AMP   = A;
    X = K*RADIUS*RADIUS/(2.0*R);
    Y = K*RADIUS;
                                                                    390
```

```
    for (I = 0; I < N; I++) {
      ARG = T + I*SAMPL;
      /*  WORK[I] += AMP*(1.0/R)*cos(modf_2PI(X -
          Y*cos(ARG - ANGLE))); */
      WORK[I] += AMP*cos(modf_2PI(X - Y*cos(ARG - ANGLE)));
      SIGNAL[I] = WORK[I];
    }
    L += 1;
    RADIUS = 0;
  }                                                          400
  free(WORK);

}

/* time_frequency_energy_map: Constructs the training
   signals and their expansions into some dictionary of orthonormal
   wavelet bases or local trigonometric bases. The resulting
   "time-frequency energy map" for each class, and the basis
   coordinates of each training signal are stored in files.
                                                              410
   We have:

   N = The length of the training signals.(Power of 2)

   L = The depth of expansion in the dictionary tree of bases.

   NS = The number of training signals in each class.

   makesignal = Function that constructs a signal
   by some sampling procedure.                                420

   expand = Function that expands the signal into
   a dictionary of the desired orthonormal basis functions. */

void time_frequency_energy_map(int N, int L, int NS, FILE *TF1,
                               FILE *TF2, FILE *SIG1, FILE *SIG2)
{
  double LOCENERGY_1, LOCENERGY_2, CLASSNORMP_1,
    CLASSNORMP_2,*WORK_1, *WORK_2,
    *GAMMA_1, *GAMMA_2;                                      430

  int LEVEL, BLOCK, POS, K, J;

  GAMMA_1 = (double *)calloc(N*(L+1), sizeof(double));
  GAMMA_2 = (double *)calloc(N*(L+1), sizeof(double));
  WORK_1 = (double *)calloc(N*(L+1), sizeof(double));
  WORK_2 = (double *)calloc(N*(L+1), sizeof(double));
  LOCENERGY_1  = 0.0;
  LOCENERGY_2  = 0.0;
```

```
CLASSNORMP_1 = 0.0;                                                    440
CLASSNORMP_2 = 0.0;
fprintf(SIG1, "%d\n", N);
fprintf(SIG1, "%d\n", L);
fprintf(SIG1, "%d\n", NS);
fprintf(SIG2, "%d\n", N);
fprintf(SIG2, "%d\n", L);
fprintf(SIG2, "%d\n", NS);

for (J = 1; J <= NS; J++) {
  makesignal(WORK_1, N, K_1, A_1, CLASS_1);                            450
  makesignal(WORK_2, N, K_2, A_2, CLASS_2);

  /* make_samples_of_1(WORK_1, N*(L+1));

  make_samples_of_oscillating_fcn(WORK_2, N*(L+1), 32); */

  CLASSNORMP_1 += lp_norm_p(WORK_1, NULL, N, 2.0);
  CLASSNORMP_2 += lp_norm_p(WORK_2, NULL, N, 2.0);
  expand(WORK_1, N, L);
  expand(WORK_2, N, L);                                                460

  for (K = 0; K < N*(L+1); K++) {
    fprintf(SIG1, "%le\n", WORK_1[K]);
    fprintf(SIG2, "%le\n", WORK_2[K]);
  }

 for (LEVEL = 0; LEVEL <= L; LEVEL++) {
   for (BLOCK = 0; BLOCK < (1<<LEVEL); BLOCK++) {
     for (POS = 0; POS < abtlength(N, LEVEL); POS++) {
                                                                       470
       LOCENERGY_1 =
         (*(WORK_1 + abtblock(N,LEVEL,BLOCK) + POS))*
         (*(WORK_1 + abtblock(N,LEVEL,BLOCK) + POS));

       LOCENERGY_2 =
         (*(WORK_2 + abtblock(N,LEVEL,BLOCK) + POS))*
         (*(WORK_2 + abtblock(N,LEVEL,BLOCK) + POS));

       *(GAMMA_1 + abtblock(N,LEVEL,BLOCK) + POS)
         += LOCENERGY_1;                                               480
       *(GAMMA_2 + abtblock(N,LEVEL,BLOCK) + POS)
         += LOCENERGY_2;

       if (J == NS) {
         *(GAMMA_1 + abtblock(N,LEVEL,BLOCK) + POS)
           /= CLASSNORMP_1;
         *(GAMMA_2 + abtblock(N,LEVEL,BLOCK) + POS)
           /= CLASSNORMP_2;
```

```
        }
       }
      }
     }
    }
    fprintf(TF1, "%d\n", N);
    fprintf(TF1, "%d\n", L);
    fprintf(TF2, "%d\n", N);
    fprintf(TF2, "%d\n", L);

    for (K = 0; K < N*(L+1); K++) {
      fprintf(TF1, "%le\n", GAMMA_1[K]);
      fprintf(TF2, "%le\n", GAMMA_2[K]);

    }
    free(WORK_1);
    free(WORK_2);
    free(GAMMA_1);
    free(GAMMA_2);
    fclose(TF1);
    fclose(TF2);
    fclose(SIG1);
    fclose(SIG2);
}

/* ldb_search_and_sort: Do a LDB search and sorting procedure
   on two given binary tree arrays X and Y of time-frequency
   energy maps resulting from a collection of wavelet packet
   dictionarys or local cosine/sine dictionarys of training
   signals belonging to two distinct classes, maximizing some
   given discriminant measure on the time-frequency energy maps.

   We have:

   TF_1 = Pointer to the file of time-frequency energy
   maps resulting from the set of expanded training signals
   belonging to a certain class, arranged level by level,
   block by block.

   TF_2 = Pointer to the file of  time-frequency energy
   maps resulting from the set of expanded training signals
   belonging to some other class, arranged level by level,
   block by block.

   OUT = Pointer to file of output data.

   N = The length of each training signal.
```

```
   L = The depth of expansion of each signal in
   the dictionarys. */
                                                                    540
void ldb_search_and_sort(FILE *TF_1, FILE *TF_2, FILE *OUT)
{

  int LEVEL, BLOCK, BLOCKLEN, POS, PARENT,
    NODE, CHILDNODE, N, L, K;

  double DELTA, DELTACHI, *WORK_1, *WORK_2;

  struct basisnode **BASIS, **TEMP;
                                                                    550
  fscanf(TF_1, "%d", &N);
  fscanf(TF_1, "%d", &L);
  fscanf(TF_2, "%d", &N);  /* Advancement of FILE pointer */
  fscanf(TF_2, "%d", &L);  /* Advancement of FILE pointer */

  WORK_1 = (double *)calloc(N*(L+1), sizeof(double));
  WORK_2 = (double *)calloc(N*(L+1), sizeof(double));
  TEMP = (struct basisnode **)calloc(1<<(L+1),
                                     sizeof(struct basisnode *));
  BASIS = (struct basisnode **)calloc(N,                            560
                                      sizeof(struct basisnode *));

  for (K = 0; K < N*(L+1); K++) {
    fscanf(TF_1, "%lf", &WORK_1[K]);
    fscanf(TF_2, "%lf", &WORK_2[K]);
  }
  BLOCKLEN = abtlength(N,L);

  for (BLOCK = 0; BLOCK < (1<<L); BLOCK++) {
    NODE = (1<<L) - 1 + BLOCK;                                      570
    DELTA =  s_measure(WORK_1 + L*N + BLOCK*BLOCKLEN,
                       WORK_2 + L*N + BLOCK*BLOCKLEN,
                       BLOCKLEN);
    TEMP[NODE] = makebasisnode(L, BLOCK, 0, DELTA, YES);
  }

  for (LEVEL = L-1; LEVEL >= 0; LEVEL--) {
    BLOCKLEN = abtlength(N,LEVEL);

    for (BLOCK = 0; BLOCK < (1<<LEVEL); BLOCK++) {                  580
      PARENT = abtblock(N, LEVEL, BLOCK);
      NODE = (1<<LEVEL) - 1 + BLOCK;
      CHILDNODE = 2*NODE + 1;
      DELTA = s_measure(WORK_1+PARENT,
                        WORK_2+PARENT, BLOCKLEN);
      DELTACHI = TEMP[CHILDNODE]->COST +
```

```
      TEMP[CHILDNODE + 1]->COST;
    if (DELTA >= DELTACHI) {
      set_tag_to_zero(TEMP, L, LEVEL, BLOCK);
      TEMP[NODE] = makebasisnode(LEVEL, BLOCK, 0,                    590
                                 DELTA, YES);
    }
    else
      TEMP[NODE] = makebasisnode(LEVEL, BLOCK, 0,
                                 DELTACHI, NO);
  }
}
K = 0;

for (NODE = 0; NODE < ((1<<(L+1))-1); NODE++) {                      600
  if (K < N) {
    if (TEMP[NODE]->TAG == YES) {
      LEVEL = TEMP[NODE]->LEVEL;
      BLOCK = TEMP[NODE]->BLOCK;
      for (POS = 0; POS < (N>>LEVEL); POS++) {
        DELTA = s_measure(WORK_1 +
                          abtblock(N, LEVEL, BLOCK) + POS,
                          WORK_2 +
                          abtblock(N, LEVEL, BLOCK) + POS, 1);
        BASIS[K] = makebasisnode(LEVEL, BLOCK, POS,                  610
                                 DELTA, YES);
        K += 1;
      }
    }
  }
}
my_qsort((void **)BASIS, 0, N-1,
         (int (*)(void *, void *))comp_basisnode_by_cost);
fprintf(OUT, "%d\n%d\n", N, L);
                                                                     620
for (K = 0; K < N; K++)
  fprintf(OUT, "%d %d %d %le\n\n",
          BASIS[K]->LEVEL, BASIS[K]->BLOCK,
          BASIS[K]->POS, BASIS[K]->COST);

free(WORK_1);
free(WORK_2);
freebasisnodes(TEMP, 1<<(L+1));
freebasisnodes(BASIS, N);
fclose(TF_1);                                                        630
fclose(TF_2);
fclose(OUT);

}
```

```
/* make2bestcoordinates: Make a set of signals from each class
   and write out the coordinates of the two most discriminating
   LDB basis elements.

   We have:                                                      640

   NS = The number of signals to be generated.

   LDB = Pointer to file containing the LDB data.

   CL1C1 = Pointer to file containing the coordinates of the
   best LDB basis element of CLASS 1.

   CL1C2 = Pointer to file containing the coordinates of the
   second best LDB basis element of CLASS 1.                     650

   CL2C1 = Pointer to file containing the coordinates of the
   best LDB basis element of CLASS 2.

   CL2C2 = Pointer to file containing the coordinates of the
   second best LDB basis element of CLASS 2.   */

void make2bestcoordinates(int NS, FILE *LDB, FILE *CL1C1,
                          FILE *CL1C2, FILE *CL2C1, FILE *CL2C2)
{                                                                660
  double *SIGNAL_1, *SIGNAL_2, XCOORDCL1, YCOORDCL1,
    XCOORDCL2, YCOORDCL2, NORMP1, NORMP2, DISC;

  struct basisnode **BASIS;

  int N, L, LEVEL, BLOCK, POS, K, J;

  fscanf(LDB, "%d", &N);
  fscanf(LDB, "%d", &L);
  BASIS = (struct basisnode **)calloc(2, sizeof(struct basisnode *));   670
  SIGNAL_1 = (double *)calloc(N*(L+1), sizeof(double));
  SIGNAL_2 = (double *)calloc(N*(L+1), sizeof(double));

  for (K = 0; K < 2; K++) {
    if (K < N) {
      fscanf(LDB, "%d", &LEVEL);
      fscanf(LDB, "%d", &BLOCK);
      fscanf(LDB, "%d", &POS);
      fscanf(LDB, "%lf", &DISC);
      BASIS[K] = makebasisnode(LEVEL, BLOCK, POS, DISC, 1);     680
    }
  }
  fprintf(CL1C1, "%d\n", NS);
  fprintf(CL1C2, "%d\n", NS);
```

```
  fprintf(CL2C1, "%d\n", NS);
  fprintf(CL2C2, "%d\n", NS);

  for (J = 0; J < NS; J++) {
    makesignal(SIGNAL_1, N, K_1, A_1, CLASS_1);
    makesignal(SIGNAL_2, N, K_2, A_2, CLASS_2);
    NORMP1 = lp_norm_p(SIGNAL_1, NULL, N, 2.0);
    NORMP2 = lp_norm_p(SIGNAL_2, NULL, N, 2.0);
    expand(SIGNAL_1, N, L);
    expand(SIGNAL_2, N, L);

    XCOORDCL1 = *(SIGNAL_1 +
                  abtblock(N, BASIS[0]->LEVEL, BASIS[0]->BLOCK) +
                  BASIS[0]->POS);
    XCOORDCL2 = *(SIGNAL_2 +
                  abtblock(N, BASIS[0]->LEVEL, BASIS[0]->BLOCK) +
                  BASIS[0]->POS);
    YCOORDCL1 = *(SIGNAL_1 +
                  abtblock(N, BASIS[1]->LEVEL, BASIS[1]->BLOCK) +
                  BASIS[1]->POS);
    YCOORDCL2 = *(SIGNAL_2 +
                  abtblock(N, BASIS[1]->LEVEL, BASIS[1]->BLOCK) +
                  BASIS[1]->POS);
    fprintf(CL1C1, "%lf\n", XCOORDCL1);
    fprintf(CL1C2, "%lf\n", YCOORDCL1);
    fprintf(CL2C1, "%lf\n", XCOORDCL2);
    fprintf(CL2C2, "%lf\n", YCOORDCL2);
  }
  freebasisnodes(BASIS, 2);
  free(SIGNAL_1);
  free(SIGNAL_2);
  fclose(LDB);
  fclose(CL1C1);
  fclose(CL1C2);
  fclose(CL2C1);
  fclose(CL2C2);
}

/* Write a signal of each class to a file for plotting. */

void write_signals_to_file(int N)

{
  int K;

  double *WORK1, *WORK2;

  FILE *P1, *P2;
```

```
  if(( P1 = fopen("/users/stud/eirikf/programs/hovedfag/idlplot/
  signalplot/signal1","w"))
     == NULL) {
    printf("ERROR in opening signal1\n");
    exit(1);
  }
  if(( P2 = fopen("/users/stud/eirikf/programs/hovedfag/idlplot/
  signalplot/signal2","w"))
     == NULL) {
    printf("ERROR in opening F2\n");
    exit(1);
  }
  WORK1 = (double *)calloc(N, sizeof(double));
  WORK2 = (double *)calloc(N, sizeof(double));
   makesignal(WORK1, N, K_1, A_1, CLASS_1);
   makesignal(WORK2, N, K_2, A_2, CLASS_2);
   fprintf(P1, "%d\n", N);
   fprintf(P2, "%d\n", N);

   for (K = 0; K < N; K++) {
     fprintf(P1, "%le\n", WORK1[K]);
     fprintf(P2, "%le\n", WORK2[K]);
   }
   free(WORK1);
   free(WORK2);
   fclose(P1);
   fclose(P2);
}

/* Compute the best decision surface of our two classes of signals
   that can be obtained from the shape of a "hyper-sphere".

    M is the maximum number of coordinates to be used in constructing
    the "hypersphere".

    1/1<<POWER is the step size in the search grid.

    LIMIT is the upper search limit for the best radius.

    LDB is pointer to file containing the coordinates
    of the most discriminating basis elements.

    BESTSPHERE is pointer to file of output data.

    OPTION is a "switch" to reverse the definition of
    the classifier if needed. */

void find_decision_surface(int M, int POWER, double LIMIT,
                           FILE *LDB, FILE *BESTSPHERE, int OPTION)
```

```
{
  int N, L, NS, LEVEL, BLOCK, POS, MISPLACED,
    LEASTMISPLACED, BESTDIMENSION, COUNT, LENGTH, J, K, I;

  double *WORK1, *WORK2, RADIUS, BESTRADIUS, DELTA,
    STEP, *TRAINPOINT1, *TRAINPOINT2, DISTANCE1, DISTANCE2,
    MISCLASSERROR;
                                                                    790
  struct basisnode **LDBDATA;

  FILE *SIG1, *SIG2;

  SIG1 = fopen("/users/stud/eirikf/work/timefrequency/sig1","r");
  SIG2 = fopen("/users/stud/eirikf/work/timefrequency/sig2","r");
  fscanf(LDB, "%d", &N);
  fscanf(LDB, "%d", &L);
  fscanf(SIG1, "%d", &N);
  fscanf(SIG1, "%d", &L);                                           800
  fscanf(SIG1, "%d", &NS);
  fscanf(SIG2, "%d", &N);
  fscanf(SIG2, "%d", &L);
  fscanf(SIG2, "%d", &NS);
  LENGTH = NS*N*(L+1);
  LDBDATA = (struct basisnode **)calloc(M, sizeof(struct basisnode *));
  WORK1 = (double *)calloc(LENGTH, sizeof(double));
  WORK2 = (double *)calloc(LENGTH, sizeof(double));

  for (I = 0; I < LENGTH; I++) {                                    810
    fscanf(SIG1, "%lf", &WORK1[I]);
    fscanf(SIG2, "%lf", &WORK2[I]);
  }
  TRAINPOINT1 = (double *)calloc(M, sizeof(double));
  TRAINPOINT2 = (double *)calloc(M, sizeof(double));

  for (K = 0; K < M; K++) {
    fscanf(LDB, "%d", &LEVEL);
    fscanf(LDB, "%d", &BLOCK);
    fscanf(LDB, "%d", &POS);                                        820
    fscanf(LDB, "%lf", &DELTA);
    LDBDATA[K] = makebasisnode(LEVEL, BLOCK, POS, DELTA, 0);
  }

  STEP = 1.0/(1<<POWER);
  LEASTMISPLACED = 2*NS;

  for (J = 2; J <= M; J++) {
    COUNT = 1;
    while (STEP*COUNT <= LIMIT) {                                   830
      MISPLACED = 0;
```

```
    RADIUS = STEP*COUNT;

    for (K = 0; K < NS; K++) {
      DISTANCE1 = 0;
      DISTANCE2 = 0;

      for (I = 0; I < J; I++) {
        LEVEL = LDBDATA[I]->LEVEL;
        BLOCK = LDBDATA[I]->BLOCK;                                    840
        POS   = LDBDATA[I]->POS;
        TRAINPOINT1[I] =
          *(WORK1 + K*N*(L+1) +
            abtblock(N,LEVEL,BLOCK) + POS);
        TRAINPOINT2[I] =
          *(WORK2 + K*N*(L+1) +
            abtblock(N,LEVEL,BLOCK) + POS);
        TRAINPOINT1[I] *= TRAINPOINT1[I];
        TRAINPOINT2[I] *= TRAINPOINT2[I];
        DISTANCE1 += TRAINPOINT1[I];                                  850
        DISTANCE2 += TRAINPOINT2[I];
      }
      DISTANCE1 = sqrt(DISTANCE1);
      DISTANCE2 = sqrt(DISTANCE2);
      if (OPTION == 0) {
        if (DISTANCE1 < RADIUS)
          MISPLACED += 1;
        if (DISTANCE2 > RADIUS)
          MISPLACED += 1;
      }                                                               860
      else {
        if (DISTANCE1 > RADIUS)
          MISPLACED += 1;
        if (DISTANCE2 < RADIUS)
          MISPLACED += 1;
      }
    }
    if (MISPLACED < LEASTMISPLACED) {
      LEASTMISPLACED = MISPLACED;
      BESTDIMENSION = J;                                              870
      BESTRADIUS = RADIUS;
    }
    COUNT += 1;
  }
}
MISCLASSERROR = (1.0*LEASTMISPLACED)/(2.0*NS);
fprintf(BESTSPHERE, "%d\n", BESTDIMENSION);
fprintf(BESTSPHERE, "%lf\n", MISCLASSERROR);
fprintf(BESTSPHERE, "%lf\n", BESTRADIUS);
free(WORK1);                                                          880
```

```
  free(WORK2);
  free(TRAINPOINT1);
  free(TRAINPOINT2);
  freebasisnodes(LDBDATA, M);
  fclose(LDB);
  fclose(BESTSPHERE);
  fclose(SIG1);
  fclose(SIG2);
}

/* Given a decision surface to separate our two classes of
   signals, generate a collection of test signals and measure
   the misclassification rate on this collection using
   the given surface. Also compute the misclassification rate
   on the collection of training signals.

   NS is the number of test signals to be generated from each class.

   BESTSPHERE is a pointer to file containing the data
   determining the best classifier using a hyperspherical surface.

   LDB is a pointer to file containing the coordinates of
   the most discriminating basis functions.

   DATA is a pointer to the output file.

   OPTION is a "switch" to reverse the definition of the
   classifier. */

void test_decision_surface(int NS, FILE *BESTSPHERE,
                    FILE *LDB, FILE *DATA, int OPTION)
{
  int LEVEL, BLOCK, POS, BESTDIMENSION, MISPLACED,
    N, L, J, K, I;

  double *WORK1, *WORK2, BESTRADIUS, *TESTPOINT1,
    *TESTPOINT2, MCRTEST, MCRTRAIN, DISTANCE1,
    DISTANCE2, DELTA;

  struct basisnode **LDBDATA;

  fscanf(BESTSPHERE, "%d", &BESTDIMENSION);
  fscanf(BESTSPHERE, "%lf", &MCRTRAIN);
  fscanf(BESTSPHERE, "%lf", &BESTRADIUS);
  LDBDATA = (struct basisnode **)calloc(BESTDIMENSION,
                                  sizeof(struct basisnode));
  fscanf(LDB, "%d", &N);
  fscanf(LDB, "%d", &L);
  WORK1 = (double *)calloc(N*(L+1), sizeof(double));
```

```
  WORK2 = (double *)calloc(N*(L+1), sizeof(double));
  TESTPOINT1 = (double *)calloc(BESTDIMENSION, sizeof(double));
  TESTPOINT2 = (double *)calloc(BESTDIMENSION, sizeof(double));

  for (K = 0; K < BESTDIMENSION; K++) {
    fscanf(LDB, "%d", &LEVEL);
    fscanf(LDB, "%d", &BLOCK);
    fscanf(LDB, "%d", &POS);
    fscanf(LDB, "%lf", &DELTA);
    LDBDATA[K] = makebasisnode(LEVEL, BLOCK, POS, DELTA, 0);
  }
  MISPLACED = 0;

  for (J = 1; J <= NS; J++) {
    makesignal(WORK1, N, K_1, A_1, CLASS_1);
    makesignal(WORK2, N, K_2, A_2, CLASS_2);
    expand(WORK1, N, L);
    expand(WORK2, N, L);
    DISTANCE1 = 0;
    DISTANCE2 = 0;

    for (I = 0; I < BESTDIMENSION; I++) {
      LEVEL = LDBDATA[I]->LEVEL;
      BLOCK = LDBDATA[I]->BLOCK;
      POS   = LDBDATA[I]->POS;
      TESTPOINT1[I] =
        *(WORK1 + abtblock(N,LEVEL,BLOCK) + POS);
      TESTPOINT2[I] =
        *(WORK2 + abtblock(N,LEVEL,BLOCK) + POS);
      TESTPOINT1[I] *= TESTPOINT1[I];
      TESTPOINT2[I] *= TESTPOINT2[I];
      DISTANCE1 += TESTPOINT1[I];
      DISTANCE2 += TESTPOINT2[I];
    }
    DISTANCE1 = sqrt(DISTANCE1);
    DISTANCE2 = sqrt(DISTANCE2);
    if (OPTION == 0) {
      if (DISTANCE1 < BESTRADIUS)
        MISPLACED += 1;
      if (DISTANCE2 > BESTRADIUS)
        MISPLACED += 1;
    }
    else {
      if (DISTANCE1 > BESTRADIUS)
        MISPLACED += 1;
      if (DISTANCE2 < BESTRADIUS)
        MISPLACED += 1;
    }
  }
```

```
  MCRTEST = (1.0*MISPLACED)/(2.0*NS);
  fprintf(DATA, "%d\n", NS);                                          980
  fprintf(DATA, "%d\n", BESTDIMENSION);
  fprintf(DATA, "%lf\n", MCRTRAIN);
  fprintf(DATA, "%lf\n", MCRTEST);
  fprintf(DATA, "%lf\n", BESTRADIUS);
  free(WORK1);
  free(WORK2);
  free(TESTPOINT1);
  free(TESTPOINT2);
  freebasisnodes(LDBDATA, BESTDIMENSION);
  fclose(LDB);                                                        990
  fclose(BESTSPHERE);
  fclose(DATA);
}
```

3. Author's Maple code

```
# Compute the SO2(R) elements from the filter coefficients. #

lagevinkler :=
   proc (k) ;
      for i from 1 to N  do
         if i < N then  x(i) := evalf ( h(i−1,i−1) / h(i−1,i) ) fi ;
         for j from i to 2 ∗ N − i do
            if type ( i + j, odd ) then
           h(i,j) := evalf ( − x(i) ∗ h(i−1,j+1) + h(i−1,j) )
       else
            h(i,j) := evalf ( x(i) ∗ h(i−1,j−1) + h(i−1,j) )
              fi ;
         od ;
      if i = N then x(N) := evalf ( − h(N−1,N) / h(N−1,N−1) )
      fi ;
      od ;
      for i  from 1 to N do print ( i, x(i) )
      od ;
   end ;

# Compute the polynomials P{n}(m) of degree n that result from lowpass
filtering of the monomials m^{n}. #


f := proc( j,i,m ) ;

if m = 0 then RETURN(1/10) else
RETURN( ( (j+2∗i)/10)^m ) fi ;
end ;


z(1) :=   .4122865951 ;
z(2) :=     1.831178514 ;
z(3) :=   −.1058894200 ;
z(4) :=   7.508378888 ;
z(5) :=    −.02083494630 ;
z(6) :=   −.04396989341  ;
z(7) :=   −.004543409641  ;
z(8) :=   15.03636438  ;
z(9) :=   −.009120773147  ;
z(10) := 4.100416655 ;
z(11) := 15.61099604 ;
z(12) := 11.59905847 ;
z(13) := 37.56973541 ;
z(14) := 197.1316159 ;
z(15) := −.0004543371650 ;
```

```
N := 3 ;

javel := proc( l )


for m from 0 to 6 do
for i from −4 to 4 do
for j from 0 to 2∗N−1 do a(m,0,j,i) := evalf( f(j,i,m) ) od ;
for k from 1 to N do
for j from k−1 to 2∗N−k do
if type ( k+j,odd ) then
a(m,k,j,i) := evalf( −z(k)∗a(m,k−1,j+1,i)+a(m,k−1,j,i) ) else
a(m,k,j,i) := evalf(  z(k)∗a(m,k−1,j−1,i)+a(m,k−1,j,i) )
fi ;
od ;
od ;
od ;
od ;


p0 := a(0,N,N−1,0) ;

p1 := n −> B1∗n + C1 ;

p2 := n −> B2∗n^2 + C2∗n + D2 ;

p3 := n −> B3∗n^3 + C3∗n^2 +D3∗n + E3 ;

p4 := n −> B4∗n^4 + C4∗n^3 + D4∗n^2 + E4∗n + F4 ;

s := solve( { p1(N−1)=a(1,N,N−1,0),
p1(N+1)=a(1,N,N−1,1) },{ B1,C1 } ) ;
assign( s ) ;

s := solve( { p2(N−1)=a(2,N,N−1,0),
p2(N+1)=a(2,N,N−1,1),
p2(N+3)=a(2,N,N−1,2) },{ B2,C2,D2 } ) ;
assign( s ) ;

s := solve( { p3(N−3)=a(3,N,N−1,−1),
p3(N−1)=a(3,N,N−1,0),p3(N+1)=a(3,N,N−1,1),
p3(N+3)=a(3,N,N−1,2) },{ B3,C3,D3,E3 } ) ;
assign( s ) ;

s := solve( { p4(N−5)=a(4,N,N−1,−2),
p4(N−3)=a(4,N,N−1,−1),p4(N−1)=a(4,N,N−1,0),
p4(N+1)=a(4,N,N−1,1),p4(N+3)=a(4,N,N−1,2) },{ B4,C4,D4,E4,F4 } ) ;
assign( s ) ;
```

```
print( p0 ) ;

end ;                                                                    100


# Compute the row vectors of the edge matrices by claiming
some vanishing moments on the highpass edge coefficients and
certain polynomial conditions on the lowpass edge coefficients ensuring
polynomials of degree up to n mapping continuously to polynomials of
the same degree. #

with( linalg ) ;
                                                                         110
jafs := proc( o )

v(0) := array( 1..5,[a(0,0,0,0),a(0,1,1,0),a(0,2,2,0),a(0,3,3,0),0] ) ;
v(1) := array( 1..5,[a(1,0,0,0),a(1,1,1,0),a(1,2,2,0),a(1,3,3,0),0] ) ;
v(2) := array( 1..5,[a(2,0,0,0),a(2,1,1,0),a(2,2,2,0),a(2,3,3,0),0] ) ;
v(3) := array( 1..5,[a(3,0,0,0),a(3,1,1,0),a(3,2,2,0),a(3,3,3,0),0] ) ;
v(4) := array( 1..5,[a(4,1,0,0),a(4,2,1,0),a(4,3,2,0),a(4,4,3,0),
a(4,5,4,0)] ) ;

                                                                         120
c0 := array( 1..5,[] ) ;
c0[5] := 0 ;

d0 := array( 1..5,[] ) ;
d0[4] :=0 ;
d0[5] := 0 ;

c1 := array( 1..5,[] ) ;

d1 := array( 1..5,[] ) ;                                                 130
d1[5] := 0 ;

d2 := array( 1..5,[] ) ;

s := solve( { dotprod( c0,v(0) ) = p0, dotprod( c0,v(1) ) = p1(1),
dotprod( c0,v(2) ) = p2(1), dotprod( c0,v(3) ) = p3(1) },
{ c0[1],c0[2],c0[3],c0[4] } ) ;
assign( s ) ;

s := solve( { dotprod( d0,v(1) ) = 0, dotprod( d0,v(0) ) = 0 },          140
{ d0[1],d0[2] } ) ;
assign( s ) ;

s := solve( { dotprod( c1,v(0) ) = p0, dotprod( c1,v(1) ) = p1(3),
dotprod( c1,v(2) ) = p2(3), dotprod( c1,v(3) ) = p3(3) },
```

```
{ c1[1],c1[2],c1[3],c1[4] } ) ;
assign( s ) ;

s := solve( { dotprod( d1,v(1) ) = 0, dotprod( d1,v(2) ) = 0,
dotprod( d1,v(0) ) = 0 },
{ d1[1],d1[2],d1[3] } ) ;
assign( s ) ;


i0 := -10.5 ;
j0 := -10.5 ;
l0 := -10.5 ;

d0[3] := i0 ;
c1[5] := j0 ;
d1[4] := l0 ;

A := evalm( array( 1..5,1..5, [(1,1)= d0[1],(1,2)=d0[2],(1,3)=d0[3],
(1,4)=d0[4],(1,5)=d0[5],(2,1)=c0[1],(2,2)=c0[2],(2,3)=c0[3],(2,4)=c0[4],
(2,5)=c0[5],(3,1)=d1[1],(3,2)=d1[2],(3,3)=d1[3],(3,4)=d1[4],(3,5)=d1[5],
(4,1)=c1[1],(4,2)=c1[2],(4,3)=c1[3],(4,4)=c1[4],(4,5)=c1[5],(5,1)=0,
(5,2)=0,(5,3)=0,(5,4)=0,(5,5)=1 ] ) ) ;

B := evalm( inverse( A ) ) ;

z := evalf( norm( B,1 ) + norm( A,1 ) ) ;

for i from i0 by 1 to 10.5 do
for j from j0 by 1 to 10.5 do
for l from j0 by 1 to 10.5 do

d0[3] := i ;
c1[5] := j ;
d1[4] := l ;

A := evalm( array( 1..5,1..5, [(1,1)= d0[1],(1,2)=d0[2],(1,3)=d0[3],
(1,4)=d0[4],(1,5)=d0[5],(2,1)=c0[1],(2,2)=c0[2],(2,3)=c0[3],(2,4)=c0[4],
(2,5)=c0[5],(3,1)=d1[1],(3,2)=d1[2],(3,3)=d1[3],(3,4)=d1[4],(3,5)=d1[5],
(4,1)=c1[1],(4,2)=c1[2],(4,3)=c1[3],(4,4)=c1[4],(4,5)=c1[5],(5,1)=0,
(5,2)=0,(5,3)=0,(5,4)=0,(5,5)=1 ] ) ) ;

B := evalm( inverse( A ) ) ;

w := evalf( norm( B,1 ) + norm( A,1 ) ) ;

if w < z then z := w ;
i0 := i ;
j0 := j ;
l0 := l ;
```

```
fi ;
od ;
od ;
od ;

d0[3] := i0 ;                                                                    200
c1[5] := j0 ;
d1[4] := l0 ;

A := evalm( array( 1..5,1..5, [(1,1)= d0[1],(1,2)=d0[2],(1,3)=d0[3],
(1,4)=d0[4],(1,5)=d0[5],(2,1)=c0[1],(2,2)=c0[2],(2,3)=c0[3],(2,4)=c0[4],
(2,5)=c0[5],(3,1)=d1[1],(3,2)=d1[2],(3,3)=d1[3],(3,4)=d1[4],(3,5)=d1[5],
(4,1)=c1[1],(4,2)=c1[2],(4,3)=c1[3],(4,4)=c1[4],(4,5)=c1[5],(5,1)=0,
(5,2)=0,(5,3)=0,(5,4)=0,(5,5)=1 ] ) ) ;

B := evalm( inverse( A ) ) ;                                                     210

print( z ) ;
print( 1,norm( A,1 ) ) ;
print( 2,norm( B,1 ) ) ;
print( 1,A ) ;
print( 2,B ) ;
print( evalm( B &* A ) ) ;



                                                                                 220

end ;
```

Bibliography

[1] I. Daubechies, *Ten Lectures on Wavelets*, Siam (1992).
[2] M.V. Wickerhauser, *Adapted Wavelet Analysis from Theory to Software*, A K Peters (1994).
[3] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press (1996).
[4] N. Saito, *Local Feature Extraction and Its Applications Using a Library of Bases*, dissertation, Yale University (1994).